Mise au point d'un formalisme syntaxique de haut niveau pour le traitement automatique des langues

Jerome Kirman

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Jérôme Kirman**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Mise au point d'un formalisme syntaxique de haut niveau pour le traitement automatique des langues

---

Date de soutenance : 4 décembre 2015

Devant la commission d'examen composée de :

Géraud Sénizergues .	Professeur des universités, LaBRI ..	Président
Bruno Courcelle	Professeur des universités, LaBRI ..	Directeur
Sylvain Salvati	Chargé de recherche, INRIA	Co-directeur
Lionel Clément	Maître de conférences, LaBRI	Co-directeur
Guy Perrier	Professeur des universités, LORIA .	Rapporteur
Denys Duchier	Professeur des universités, LIFO ...	Rapporteur
Éric de la Clergerie	Chargé de recherche, INRIA	Examinateur
Sylvain Schmitz	Maître de conférences, ENS Cachan	Examinateur

– 2015 –

Résumé La linguistique informatique a pour objet de construire un modèle formel des connaissances linguistiques, et d'en tirer des algorithmes permettant le traitement automatique des langues. Pour ce faire, elle s'appuie fréquemment sur des grammaires dites génératives, construisant des phrases valides par l'application successive de règles de réécriture. Une approche alternative, basée sur la théorie des modèles, vise à décrire la grammaticalité comme une conjonction de contraintes de bonne formation, en s'appuyant sur des liens profonds entre logique et automates pour produire des analyseurs efficaces. Notre travail se situe dans ce dernier cadre.

En s'appuyant sur plusieurs résultats existants en informatique théorique, nous proposons un outil de modélisation linguistique expressif, conçu pour faciliter l'ingénierie grammaticale. Celui-ci considère dans un premier temps la structure abstraite des énoncés, et fournit un langage logique s'appuyant sur les propriétés lexicales des mots pour caractériser avec concision l'ensemble des phrases grammaticalement correctes. Puis, dans un second temps, le lien entre ces structures abstraites et leurs représentations concrètes (en syntaxe et en sémantique) est établi par le biais de règles de linéarisation qui exploitent la logique et le lambda-calcul.

Par suite, afin de valider cette approche, nous proposons un ensemble de modélisations portant sur des phénomènes linguistiques divers, avec un intérêt particulier pour le traitement des langages présentant des phénomènes d'ordre libre (c'est-à-dire qui autorisent la permutation de certains mots ou groupes de mots dans une phrase sans affecter sa signification), ainsi que pour leur complexité algorithmique.

Title A high-level syntactic formalism for natural language processing

Abstract The goal of computational linguistics is to provide a formal account linguistical knowledge, and to produce algorithmic tools for natural language processing. Often, this is done in a so-called generative framework, where grammars describe sets of valid sentences by iteratively applying some set of rewrite rules. Another approach, based on model theory, describes instead grammaticality as a set of well-formedness logical constraints, relying on deep links between logic and automata in order to produce efficient parsers. This thesis favors the latter approach.

Making use of several existing results in theoretical computer science, we propose a tool for linguistical description that is both expressive and designed to facilitate grammar engineering. It first tackles the abstract structure of sentences, providing a logical language based on lexical properties of words in order to concisely describe the set of grammaticaly valid sentences. It then draws the link between these abstract structures and their representations (both in syntax and semantics), through the use of linearization rules that rely on logic and lambda-calculus.

Then in order to validate this proposal, we use it to model various linguistic phenomenas, ending with a specific focus on languages that include free word order phenomenas (that is, sentences which allow the free reordering of some of their words or syntagmas while keeping their meaning), and on their algorithmic complexity.

Mots-clés Linguistique informatique, Logique, Lambda-calcul, Grammaires catégorielles abstraites, Syntaxe modèle-théorique

Keywords Computational linguistics, Logic, Lambda-calculus, Abstract categorial grammars, Model-theoretic syntax

Laboratoire d'accueil Laboratoire Bordelais de Recherche en Informatique

Remerciements

Est-il besoin de le préciser ? Cette thèse n'aurait certes pas vu le jour sans la contribution et le soutien de nombreuses personnes ; et d'abord et surtout, je remercie mes encadrants.

Bruno Courcelle, pour sa bienveillante supervision durant ma thèse ; mais aussi pour sa contribution pédagogique avant même le début de celle-ci, parfois sous une forme inattendue (quel plaisir de découvrir, au détour d'une n-ième UE de projet de programmation, un sujet riche d'intérêt sur le plan de l'informatique théorique !).

Sylvain Salvati, par ses conseils et son encadrement sans faille au quotidien, a forgé cette thèse et assuré la rigueur de ses résultats. Sans sa connaissance exhaustive de la littérature et sa constante disponibilité, ce manuscrit n'aurait pu devenir ce qu'il est ; et je ne saurais le remercier assez.

Lionel Clément, qui d'une part m'a donné l'impulsion pour m'intéresser de plus près à la linguistique informatique, et qui d'autre part a largement contribué à l'amélioration du formalisme, tant par ses retours sur son utilisabilité que par ses lumières sur de nombreuses notions de linguistique qui manquaient cruellement à ma formation initiale.

À la veille de ma soutenance, je tiens également à remercier les autres membres de mon jury de thèse. Denys Duchier, pour sa lecture attentive du manuscrit, ainsi que Guy Perrier, dont les remarques détaillées m'ont permis nombre d'améliorations ; de même qu'Éric de la Clergerie et Sylvain Schmitz pour leur intérêt et leur temps.

Et pour finir, Géraud Sénizergues, dont la présence au sein du jury signifie beaucoup pour moi. Ses qualités pédagogiques m'ont donné, dans les premières années de mon parcours en informatique, l'envie et les moyens de me plonger dans la science informatique. Ses cours et TD d'informatique théorique m'ont permis de réaliser que l'informatique n'est pas qu'un outil, mais aussi un sujet passionnant par lui-même, et m'ont motivé dans la suite de mes études.

Au delà de ces appuis solides à mon parcours, je souhaite aussi remercier bien d'autres personnes dont la contribution, quoique plus informelle, reste à mes yeux importante.

Proche de mon domaine de travail, tout d'abord, l'ensemble des membres (actuels et anciens) de l'association Jeunes-Science Bordeaux, qui m'a permis de découvrir il y a bientôt vingt ans l'envers du décor des sciences en général et de l'informatique en particulier. Sans pouvoir citer tous les noms, d'excellents animateurs comme Christelle, Grégoire et Arthur, et des membres du milieu académique comme Stéphane Vincent ou David Smith ont permis à JSB de rester pour les jeunes un havre de curiosité envers les sciences, dont l'intérêt est chaque semaine et chaque année confirmé par ses fidèles adhérents.

Plus proche encore, je tiens à remercier mes collègues et co-bureaux, non seulement informaticiens mais aussi rôlistes, codeurs, geeks, et plus encore, toujours prêts à débattre sans fin des mérites comparés de LaTeX, OCaml, ou GNU/Linux ou à se perdre en spéculations sur l-a-grammaticalité d'une extraction double en français ou l'intérêt relatif d'un algorithme « polynomial » portant sur les mineurs de graphes et dont la constante se calcule en tours d'exponentielles. Alia, DraKlaW, DS, Zéhir, Lleld' et @vx parmi bien d'autres se seront reconnus.

Incontournable également pour moi, la communauté [NFOR] et l'ensemble de ses membres... dont la contribution à cette thèse est, en toute lucidité, très contestable, mais dont la sympathie et la bonne humeur m'ont aidé à garder le cap dans les moments de doute.

Et pour finir, je remercie les membres de ma famille, qui ont toujours été à mes côtés, et en particulier ma mère et ma sœur. Également, Philippe Gaborit, dont les conseils m'ont aidé et incité, il y a bientôt quatre ans, à me lancer dans l'aventure.

Merci à tous !

Table des matières

Introduction

La linguistique informatique a pour objet de construire un modèle mathématique approximatif des connaissances linguistiques, afin de pouvoir le mettre à contribution pour des tâches de traitement automatique. Elle distingue traditionnellement deux aspects complémentaires de la langue : la performance et la compétence. La performance linguistique est l'ensemble des énoncés réels produits par des locuteurs, et son étude se fait par le biais de corpus rassemblant des échantillons représentatifs des faits de langues existants. À l'inverse, la compétence linguistique caractérise la capacité d'un locuteur à produire et reconnaître des énoncés inédits, en fonction de leur adéquation avec l'ensemble des règles de bonne formation qui forment la grammaire de la langue. Cette thèse se concentre principalement sur ce dernier aspect, en proposant un outil de modélisation linguistique qui place les règles de bonne formation grammaticale au premier plan.

Un compte-rendu scientifique se doit d'être capable de prédire, et non seulement d'expliquer, un phénomène. Cependant, la tâche de proposer une explication satisfaisante à la grammaticalité (ou la non-grammaticalité) d'un énoncé ne doit pas être négligée : un fichier binaire de grande taille représentant un arbre binaire de décision peut contenir un modèle adéquat de la langue, mais il ne permet pas à un être humain d'appréhender sa structure et son fonctionnement. Ce souhait de rendre compte de la langue de façon humainement abordable motive le choix, dans cette thèse, de proposer un formalisme linguistique de haut niveau (au sens des langages de programmation) : la tâche de modélisation – l'ingénierie grammaticale – y est détachée des processus d'analyse, qui sont du ressort de l'implémentation.

Ce choix de faciliter avant tout la construction de grammaires concises et expressives est toutefois dénué de sens en l'absence d'une procédure effective pour transformer ces descriptions en langages formels effectivement décidables, si possible de manière efficace (à savoir en temps polynomial par rapport à la taille d'un énoncé). Cette procédure ne peut être réduite au statut de « détail d'implémentation », et s'appuie donc sur des outils simples mais puissants issus de l'informatique fondamentale ; principalement la logique et le lambda-calcul. Ces outils s'accompagnent en effet d'une riche littérature qui permettent de garantir l'efficacité du traitement automatique des langages résultants. L'hypothèse à laquelle cette thèse souscrit ainsi est que la structure du langage peut, à

un niveau abstrait, être représentée fidèlement par des structures hiérarchiques simples (nommément des arbres réguliers) ; et que les représentations concrètes qui en découlent (forme phonologique, sémantique) peuvent être obtenues à partir de ces structures abstraites par le biais de mécanismes de transduction relativement peu complexes.

Plan de la thèse La première partie du document présente des notions et résultats existants dans la littérature, qui ont permis ou éclairé la rédaction de ce document. Le chapitre 1 contient un état de l'art succinct décrivant, sans détails formels, plusieurs courants importants de la linguistique informatique et l'éclairage qu'ils apportent sur le reste du travail présenté. Le chapitre 2, pour sa part, introduit des notations et résultats formels préexistants qui serviront dans le reste du document.

La contribution originale de la thèse forme le reste du document. Le chapitre 3 présente le formalisme linguistique, fondé sur la logique et le lambda-calcul, qui donne son titre au manuscrit. Le chapitre 4 met ensuite ce formalisme à l'épreuve, en le confrontant à une tâche non-triviale de modélisation linguistique. Celle-ci se limitant à des langues possédant un ordre des mots canonique, le chapitre 5 introduit un mécanisme algébrique pour représenter et reconnaître des énoncés où l'ordre des mots est libre, en vue de compléter les capacités de modélisation du formalisme.

Enfin, le document comporte deux annexes : l'annexe A, liée au chapitre 3, contient un ensemble de stratégies montrant la confluence du système de réécriture permettant d'interpréter les règles de linéarisation du formalisme, ainsi qu'une preuve de normalisation forte pour ce système. Finalement, l'annexe B contient les démonstrations détaillées de plusieurs résultats de complexité algorithmique établis dans le chapitre 5, ainsi qu'un système de typage sémantique et les preuves de son bon fonctionnement, permettant d'établir la complexité de l'algorithme d'analyse qui conclut le chapitre.

Chapitre 1

État de l'art

Toute approche scientifique de la langue requiert un certain degré de formalisation, attribuant au langage et à ses énoncés une structure. C'est la préoccupation essentielle de la linguistique informatique.

Un des courants les plus célèbres de cette discipline est celui fondé par Chomsky [1957], proposant originellement la notion de grammaire générative, basée sur les grammaires hors-contexte. La structure de la phrase y est hiérarchique, et correspond à l'arbre de dérivation : chaque composant syntaxique se décompose en plusieurs éléments, jusqu'à parvenir à des mots (indivisibles) qui correspondent aux feuilles de l'arbre. Ce courant a servi de point de départ à de nombreuses théories : la théorie X-barre [Chomsky, 1970], généralisant la structure des propositions ; principes et paramètres [Chomsky, 1981], qui vise à classifier les différences et similarités des langages naturels ; gouvernement et liage (*ibid*), établissant des principes structurels plus précis sur les structures syntaxiques (notamment vis-à-vis du mouvement et des dépendances) ; et enfin le programme minimaliste [Chomsky, 1996], recherchant l'économie des descriptions scientifiques de la langue. Les grammaires nées de ce cadre, dites transformationnelles, vont usuellement au delà des grammaires hors-contexte en termes d'expressivité, mais continuent de représenter la structure des énoncés comme une hiérarchie de composants.

Parallèlement, les grammaires de dépendances [c Mel'ˇ cuk, 1979], trouvant leur origine dans les travaux de Tesnière [1959], représentent également les différents aspects d'un énoncé sous la forme d'une hiérarchie de composants, tour à tour régissants et subordonnés. Il en résulte également que la structure d'un énoncé est un arbre, que l'analyse d'une phrase revient à construire. Bien que la tradition des grammaires de dépendances soit, relativement au modèle génératif, davantage centrée sur la modélisation de la structure des énoncés que sur l'analyse et la production concrète de ces derniers, une riche littérature sur l'analyse en dépendance (dont Kübler *et al.* [2009] donne un large aperçu) permet de relier formellement hiérarchies de constituants et énoncés concrets.

1.1 La syntaxe par la théorie des modèles

Le paradigme dominant dans le traitement de la sémantique, issu de Montague [1974], est le traitement du sens des énoncés par la théorie des modèles. Bien que moins répandue, cette tradition existe également en syntaxe. La syntaxe par la théorie des modèles, également nommée syntaxe modèle-théorique (*Model-Theoretic Syntax*, ou MTS), suivant l'approche proposée par Rogers [1996], s'appuie sur la logique formelle pour exprimer la grammaticalité comme un ensemble de contraintes, permettant de sélectionner comme syntaxiquement corrects les énoncés (vus comme des modèles) qui satisfont à ces contraintes.

Cette approche, bien qu'également basée sur la logique formelle, se distingue de l'analyse en théorie de la démonstration (incluant notamment les grammaires catégorielles, formulées comme un problème de décision en logique par Lambek [1958]), laquelle interprète directement la phrase comme un objet du domaine de la logique, et vérifie sa bonne formation par rapport aux règles internes (syntaxiques) de la logique, plutôt qu'en passant par leur interprétation (sémantique) comme c'est le cas avec MTS. L'approche de la syntaxe modèle-théorique contraste également, comme l'établit Pullum et Scholz [2001], avec le modèle traditionnel de la syntaxe générative-énumérative.

La syntaxe par la théorie des modèles s'appuie dans une large mesure sur un résultat profond d'équivalence, établi par Büchi [1960] et étendu par Rabin [1969], entre la logique monadique du second ordre (*Monadic Second-Order logic*, ou MSO) et les automates à états finis, sur une structure donnée. Ce résultat permet notamment d'établir l'équivalence entre la classe des langages reconnus par les automates ou grammaires de termes (étroitement liée à celle des langages d'arbres de dérivation des grammaires hors-contexte), et celle des langages de termes qui satisfont une certaine formule logique. Il en résulte immédiatement une procédure de reconnaissance efficace des structures qui satisfont un ensembles de contraintes logiques ; ce qui permet de modéliser la correction syntaxique directement comme un ensemble de contraintes de bonnes formation à satisfaire.

En dépit de cette approche directe de la notion de grammaticalité, le courant MTS n'a pas connu une très large adoption. L'une des raisons est que la classe des énoncés obtenus dépend du choix d'un opérateur pour transformer la structure arborescente de l'énoncé en une séquence linéaire de mots : le seul candidat évident est la concaténation naïve des feuilles de l'arbre de gauche à droite, ce qui limite la classe des langages définissables par ce biais à celle des langages hors-contextes, aujourd'hui jugés insuffisants pour l'analyse de plusieurs phénomènes linguistiques réels (voir section 1.2).

La dernière section de Cornell et Rogers [1998] évoque plusieurs réponses possibles à cette limitation, incluant l'emploi de structures support plus élaborées que des arbres ou mots [*cf.* Rogers, 1998], ou celui d'un langage logique plus puissant que MSO. Ce dernier choix présente toutefois l'inconvénient que

les logiques dont l'expressivité dépasse celle de MSO sont usuellement indécidables, même sur des structures simples. Le choix de s'appuyer sur des structures plus élaborées ouvre cependant la perspective de décrire par la logique des classes de langages de mots légèrement sensibles au contexte.

1.2 Langages légèrement sensibles au contexte

Comme l'illustre le problème de l'expressivité des formalismes MTS, la notion de langage formel est centrale en linguistique informatique : une langue est modélisée par un ensemble de mots, lui-même caractérisé par une description finie. La forme que ces descriptions peuvent adopter et la classe de langages qui en résulte sont définis par le formalisme linguistique auquel on souscrit, de même que la complexité des procédures de décision et d'analyse associés.

L'étude de la classe de langages que décrit un formalisme permet de dégager deux critères possibles d'évaluation pour celui-ci : d'une part, son adéquation à une tâche de modélisation linguistique, déterminée par la possibilité de décrire un langage capturant un phénomène linguistique particulier (ou plusieurs, simultanément) ; d'autre part sa difficulté intrinsèque, bornée par la complexité du problème de l'appartenance d'un mot à un langage de la classe dans le pire cas.

Ces critères d'évaluation demeurent critiquables : le premier critère peut s'avérer trompeur, si l'on omet de tenir compte de la distinction entre pouvoir de génération faible (l'existence d'un langage capturant un phénomène) et fort (l'existence d'une grammaire capturant explicitement la structure de ce phénomène). Le second critère, quant à lui, ne donne pas d'information sur la difficulté de l'analyse en fonction d'une description (d'une grammaire) particulière, le seul paramètre étant le mot à analyser ; pas plus qu'il ne tient compte du fait que certaines descriptions correspondant à des langages algorithmiquement difficiles peuvent ne pas apparaître en pratique dans la modélisation d'une langue réelle.

En dépit de ces réserves, une étude des classes de langages générés par de nombreux formalismes dédiés au traitement des langues et conçus à la fois pour leur adéquation et leur complexité algorithmique raisonnable conduit à observer l'existence de caractéristiques communes. Cette observation a conduit à l'émergence de la notion de classes de langages dites *légèrement sensibles au contexte* (*Mildly-Context Sensitive Languages/Grammars*, ou MCSL/G), proposée par Joshi [1985]. Bien que qualifiant originellement à la fois des formalismes linguistiques et leur langages associés, et faisant toujours l'objet de discussions [*cf.* Kallmeyer, 2010], cette notion constitue un critère appréciable pour évaluer l'adéquation et la complexité d'une classe de modèles destinés à la description des langues.

Le nom de « légèrement sensibles au contexte » provient du placement de

cette classe de langages par rapport à la hiérarchie de Chomsky [1959], qui inclut deux classes de grammaires successives, générant respectivement la classe des langages hors-contexte (grammaires de type 2) et celle des langages sensibles au contexte (grammaires de type 1). La première classe est algorithmiquement abordable, ce qui en fait un candidat souhaitable pour la modélisation du langage du point de vue du traitement automatique des langues (TAL) ; cependant, un ensemble d'exemples linguistiques (incluant notamment les dépendances croisées en série de Shieber [1985]), montrent que cette classe est inadéquate pour la modélisation de nombre de phénomènes réels. À l'inverse, la classe des langages sensibles au contexte, bien que plus riche, contient des langages pour lesquels la complexité algorithmique de l'analyse est prohibitive (polynomiale en espace).

Pour cette raison, les langages légèrement sensibles au contexte sont définis comme incluant les langages hors-contexte, et capables de capturer certains phénomènes nécessaires à la modélisation des dépendances croisées évoquées plus haut. Ils sont d'autre part limités dans leur expressivité par la propriété dite de croissance constante, liée à la semilinéarité (voir sections 5.3.1 et 2.1.4), et par la contrainte d'analyse en temps polynomial. Il est à noter que cette définition demeure ambigüe, et par conséquent, plusieurs classes de langages sont candidates au titre de légèrement sensibles au contexte ; mais cette définition demeure motivée par l'observation que de nombreux formalismes (principalement destinés à la description linguistique) ont abouti, par des mécanismes indépendants, à des classes de langages identiques ou, tout du moins étroitement reliées.

Ainsi, la classe de langages originellement envisagée pour les MCS coïncidait avec celle des grammaires d'arbres adjoints (TAG, Joshi et Schabes [1997]), des grammaires de tête (HG, Pollard [1984]), des grammaires catégorielles combinatoires (CCG, Steedman [1987]) et des grammaires linéairement indexées (LIG, Gazdar [1988] – restreignant les grammaires indexées de Aho [1968]) ; cette classe est également caractérisée par les 2-$\mathrm{MCFG}_{\mathrm{WN}}$ (voir section 2.4.1).

Une autre classe de langages, plus large, qui répond également à la définition des MCS, est celle générée par les MCFG (voir section 2.4.1). Elle coïncide avec les grammaires minimalistes (MG, Michaelis [2001]), les grammaires d'arbres adjoints multiples (MC-TAG, Joshi *et al.* [1987]), ainsi qu'avec les grammaires catégorielles abstraites du second ordre sur les chaînes (2-ACG_{str}, voir 2.4.2).

Enfin, une dernière classe candidate au statut de MCS, intermédiaire entre les deux précédentes, est formée par l'ensemble des langages générés par les $\mathrm{MCFG}_{\mathrm{WN}}$, lesquelles sont abordées plus en détail dans la section 2.4.1.

1.3 Les métagrammaires

Comme le montre la diversité des formalismes ayant le même pouvoir formel de génération, le choix d'un outil de description linguistique n'a pas nécessairement de conséquence sur l'ensemble des énoncés et phénomènes descriptibles. Les critères de choix essentiels deviennent alors la maniabilité du formalisme (l'aisance avec laquelle il permet de décrire une grammaire) et son adéquation aux phénomènes modélisés (permettant de produire des descriptions simples et concises, qui reflètent directement la structure des énoncés décrits). D'autre part, la création de grammaires à large couverture, par rapport à un corpus ou un ensemble de phénomènes linguistiques, engendre souvent une explosion combinatoire du nombre de symboles et règles présents dans la grammaire, rendant difficile sa maintenance future.

L'approche des métagrammaires (MG) constitue une réponse à ces problèmes. Celles-ci consistent en un langage de description, qui ne décrit pas directement un ensemble d'énoncés mais un ensemble de symboles et productions dans un formalisme grammatical donné (originellement TAG), lesquels décrivent à leur tour le langage cible. Cette approche, originellement proposée par Shanker et Schabes [1992], a été mise en pratique par Candito [1999] et Crabbé [2005] pour la description de larges grammaires (une version plus concise de cette dernière approche est donnée par Crabbé et Duchier [2005]). Enfin, une spécification pour un outil de description de métagrammaires extensible (XMG) est décrit par Duchier *et al.* [2005], et s'accompagne d'une implémentation concrète.

Le principe fondamental de l'approche des métagrammaires est de capturer des généralisations qui échappent au formalisme cible. Il en résulte non seulement des descriptions plus concises, mais aussi plus simples, puisque traduisant explicitement les concepts linguistiques qu'elles modélisent. En un sens, elles constituent une approche programmatique de la description grammaticale, en permettant de factoriser le « code » d'une grammaire, de disposer d'espaces de noms et de répartir l'information dans différents modules (syntaxe, sémantique, ...) dont l'interdépendance est limitée. Elles permettent également de s'abstraire de considérations de « bas niveau » liées à la définition du formalisme cible : la métagrammaire décrite par Clément et Kinyon [2003] peut ainsi s'interpréter comme une grammaire d'arbre adjoints ou comme une grammaire lexicale fonctionnelle. Citons également XMG, qui permet la compilation d'une métagrammaire vers plusieurs formalismes distincts, tout comme un programme écrit dans un langage de haut niveau peut être compris et maintenu indépendamment de la machine qui servira à l'exécuter.

1.4 Perspective

À l'image des grammaires transformationnelles et des grammaires de dépendances, le formalisme proposé dans cette thèse décrit la structure syntaxique des phrases comme des arbres. Des structures plus complexes (notamment des graphes acycliques) pourront apparaître de manière intermédiaire, pour faire le lien entre structure profonde et réalisation de surface, mais les grammaires que nous décrivons n'en feront pas explicitement usage.

La description de ces structures syntaxiques s'effectuera, à l'image de MTS, en employant la théorie des modèles. La grammaticalité sera ainsi principalement définie par le biais de contraintes logiques, qu'une structure devra satisfaire pour être acceptée. La logique monadique du second ordre sur les arbres nous servira de cadre de travail, nous permettant d'exploiter les mêmes résultats théoriques que la littérature des MTS, mais limitant l'ensemble des structures syntaxiques acceptées à un langage régulier d'arbres. Puisque l'ensemble des structures que nous visons est plus riche, le lien entre les deux sera établi par le biais d'un mécanisme de linéarisation puissant, basé sur le lambda-calcul, transformant ces structures en chaînes (côté syntaxe) ou en formules logiques représentant le sens de l'énoncé (côté sémantique). Nous génèrerons ainsi une classe de langages MCS ; ce résultat est garanti par la littérature des grammaires catégorielles abstraites, introduites plus en détail dans la section 2.4.2.

Enfin, nous séparerons clairement le processus de description d'un langage de structures syntaxiques (et ses linéarisations possibles en syntaxe et en sémantique) de son implémentation. Cette approche se veut dans l'esprit des métagrammaires, afin de mêler le moins possible les suppositions linguistiques de l'auteur de la grammaire et les détails de l'implémentation du formalisme. En outre, la formulation des contraintes logiques de bonne formation et des règles de linéarisation se fera de façon modulaire, facilitant l'ingénierie grammaticale en permettant de séparer clairement les concepts linguistiques mis en jeu.

Pour terminer, il faut également mentionner que la mise à l'épreuve du formalisme s'appuiera sur des modélisations pré-existantes de certains phénomènes linguistiques ; en particulier de la modélisation des dépendances à longue distance (*long-distance dependencies*) proposée dans la tradition des grammaires lexicales-fonctionnelles (LFG, Kaplan et Bresnan [1982]).

Chapitre 2

Notions préliminaires

Ce chapitre est dédié à l'introduction de notions et résultats issus de la littérature sur lesquels s'appuie notre contribution. Outre des notations mathématiques de base, nous rappelons les notions de logique formelle et de théorie des modèles ; puis nous présentons le lambda-calcul, et en particulier la notion de typage simple et ses propriétés ; enfin, nous décrirons plusieurs systèmes issus de la théorie des langages qui s'avèrent pertinents pour répondre à des problèmes de formalisation linguistique. Nous préciserons pour chacune de ces notions l'importance qu'elle revêt dans le cadre de cette thèse.

2.1 Définitions

2.1.1 Notations

Le sous-ensemble $\{1, \ldots, n\}$ des entiers naturels $\mathbb{N}$ est abrégé en $[n]$. Le cardinal d'un ensemble E est noté $\#(E)$, et l'ensemble des parties de cet ensemble $\mathscr{P}(E)$. Si $T = (e_1 \ldots e_n)$ est un tuple, la i-ème projection de T est notée $\pi_i(T)$, pour $i \in [n]$. Si E est un ensemble, $\vec{\mathbb{N}}^E$ est l'ensemble des vecteurs de dimension E, où pour tout vecteur v, la notation $v.a$ désigne la composante associée à l'élément a de E dans v ; $\vec{0}$ désigne le vecteur nul, et $\vec{1}_a$ le vecteur dont la composante $v.a$ vaut 1, et nul partout ailleurs. Enfin, la notation $\|v\|$ désigne la norme supremum de v définie par $\|v\| = \max(v.a)$ pour $a \in E$.

La substitution d'une variable x par un élément a dans une expression E est notée $E[x \leftarrow a]$; par contraste, la substitution sans capture introduite dans la section 2.2.2 est notée $E[a/x]$.

Dans le cadre des systèmes de réécriture, si $\to$ est une relation, $a \xrightarrow{k} b$ dénote qu'il existe k éléments $a_1 \ldots a_k$ tels que $a \to a_1$ et $a_i \to a_{i+1}$ pour chaque $i \in [k-1]$, et $a_k = b$; implicitement, $k = 0$ dénote que $a = b$. Par extension, la relation $a \xrightarrow{*} b$ dénote la clôture réflexive et transitive de $\to$.

2.1.2 Alphabets, mots, langages

Soit Σ un ensemble de symboles. Nous considérons le monoïde libre (noté Σ^*) formé à partir de la base Σ par l'opération « . », qui est donc associative et dotée d'un élément neutre noté ε.

L'ensemble Σ est appelé *alphabet*, et ses éléments *lettres* ; les éléments de Σ^* sont appelés *mots*, l'opération « . » *concaténation* et son élément neutre le *mot vide*. Un ensemble (fini ou non) de mots sur Σ^* est appelé *langage*. L'ensemble Σ^* est appelé *langage universel*. Enfin, un ensemble de langages est appelé une *classe* de langages.

Pour tout mot w de Σ^* et tout entier naturel n, w^n dénote le mot formé en concaténant n fois le mot w ; par convention, $w^0 = \varepsilon$. Pour toute lettre l de Σ et tout mot w, $|w|_l$ dénote le nombre d'occurrences de l dans w ; et la longueur d'un mot est notée $|w| = \sum_{l \in \Sigma} |w|_l$.

Enfin, l'*image de Parikh* est une fonction (notée ψ) qui associe à tout mot w construit sur un alphabet Σ un vecteur v de $\vec{\mathbb{N}}^{\Sigma}$, tel que $v.l = |w|_l$ pour tout $l \in \Sigma$. Par extension, l'image de Parikh d'un langage L est définie comme l'ensemble des images de Parikh de ses mots : $\psi(L) = \{\psi(w) \mid w \in L\}$.

2.1.3 Alphabets gradués, termes, contextes

Un *alphabet gradué* est un ensemble de symboles, accompagné d'une fonction nommée *arité* (notée ρ) qui associe à tout élément de l'ensemble un entier naturel. Dans ce contexte, nous écrirons parfois s_k pour introduire un nouveau symbole s d'arité $\rho(s) = k$.

Si $\mathcal{S}$ est un alphabet gradué, l'ensemble des *termes* sur $\mathcal{S}$ est défini inductivement de la manière suivante : étant donné n termes $t_1 \dots t_n$ sur $\mathcal{S}$ et un symbole s de $\mathcal{S}$ tel que $\rho(s) = n$, $s(t_1, \dots, t_n)$ est un terme sur $\mathcal{S}$. Si s est d'arité zéro, nous écrirons parfois s au lieu de $s()$.

Une *position* dans un terme est un mot de $(\mathbb{N})^*$. Étant donné un terme t, une position p est dite *valide* dans t si et seulement si $p = \varepsilon$ ou si $p = ip'$, $t = s(t_1 \dots t_n)$ et $i \in [n]$ et que p' est une position valide dans t_i. L'ensemble des positions valides d'un terme t est appelé le *domaine* de t (noté $\mathrm{Dom}(t)$). Le *plus petit ancêtre commun* de deux positions p et p' dans un terme désigne la position construite comme le plus long préfixe commun à p et p'.

Étant donné une position p valide dans un terme t, le *sous-terme* de t à la position p (noté $t|_p$) est défini inductivement comme : $t|_p = t$ si $p = \varepsilon$; et $t|_p = t_i|_{p'}$ si $t = s(t_1 \dots t_n)$ et $p = ip'$; la définition des termes et de la validité d'une position garantit l'existence d'un tel sous-terme. Par analogie avec les arbres, une position valide désignant un sous-terme d'arité nulle est appelée une *feuille*, et la position ε la *racine*.

Considérons maintenant un ensemble dénombrable de variables $\mathcal{X}$ (notées $x, y, z, \dots$) : un *contexte* sur $\mathcal{S}$ est un terme construit sur l'alphabet gradué

$\mathcal{S} \cup \mathcal{X}$ où tous les éléments de $\mathcal{X}$ sont d'arité nulle. Tout sous-terme $x()$ dans un contexte C est appelé une *occurrence* de x dans C. Un contexte est dit *linéaire* s'il contient au plus une occurrence de x pour tout $x \in \mathcal{X}$.

Étant donné un contexte C, son *arité* $\rho(C)$ dénote le nombre de variables distinctes ayant des occurrences dans C ; un contexte d'arité k est également dit *k-aire*. Par suite, si $X = \{x_1 \dots x_n\}$ est l'ensemble ordonné des variables d'un contexte n-aire C, $C[t_1, \dots, t_n] = C[x_i \leftarrow t_i]$ dénote le résultat de la substitution simultanée de chacune des variables x_i par t_i dans C, pour $i \in [n]$. Celle-ci est distincte de la séquence de substitutions $C[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]$, en particulier si l'un des termes t_i contient une variable x_j avec $j > i$: aucune substitution n'est alors effectuée dans le terme substitué t_i. Enfin, si C est un contexte unaire et $n \in \mathbb{N}$, l'exponentiation $C^n = C[x \leftarrow C] \dots [x \leftarrow C]$ dénote le contexte obtenu en substituant $n - 1$ fois C à x (séquentiellement) ; ainsi, $C^1 = C$ et, par abus de notation, $C^0 = x$ est le contexte trivial.

2.1.4 Ensembles semilinéaires

Étant donné un ensemble E et un tuple de vecteurs $T = (v_0, \dots, v_n)$ appartenant chacun à $\vec{\mathbb{N}}^E$, l'*ensemble linéaire* défini par T est l'ensemble de vecteurs :

$$\left\{ v_0 + \sum_{i \in [n]} k_i v_i \mid \forall i \in [n], k_i \in \mathbb{N} \right\}$$

En outre, un ensemble de vecteurs est dit *semilinéaire* s'il peut être décrit comme une union finie d'ensembles linéaires de vecteurs. L'appartenance d'un vecteur v à un ensemble semilinéaire est décidable avec une complexité en espace logarithmique par rapport à la norme supremum $\|v\|$ de v (en choisissant le bon ensemble linéaire et en soustrayant itérativement un v_i judicieusement choisi jusqu'à aboutir à v_0).

2.1.5 Logique formelle

La logique formelle permet de formuler et d'évaluer la vérité de propositions portant sur une classe particulière d'objets. Un langage logique permet d'écrire des formules, lesquelles s'appuient sur un ensemble de prédicats et de relations en lien avec la classe d'objets étudiée. Différents types de logiques offrent divers degrés d'expressivité, ce qui revêt une importance particulière lorsqu'on considère, en théorie des modèles, les ensembles d'objets satisfaisant une formule en particulier. Nous définissons ici deux types de logiques, la logique du premier ordre et la logique du second ordre monadique, sur une signature quelconque, en rappelant la syntaxe et la sémantique usuelle des formules qu'elles construisent.

Signatures, formules Soit $\mathcal{X}_1$ et $\mathcal{X}_2$ deux ensembles dénombrables de variables, respectivement dites du *premier ordre* et du *second ordre*, et notées $x, y, z, \ldots$ et $X, Y, Z, \ldots$. Une *signature* est un ensemble de symboles nommés *prédicats* ou *relations* (notés $P, Q, R, \ldots$), munis de deux arités ρ_1 et ρ_2. Ces arités correspondent respectivement aux nombres de variables du premier et du second ordre dont dépend un prédicat. Nous emploierons souvent dans la suite le terme de *prédicat* par défaut, en réservant le terme de *relation* pour des prédicats dont l'arité est supérieure ou égale à deux.

Dans le cas général, une signature peut également contenir des *termes*, interprétés comme des constantes, ainsi que des *fonctions* s'appliquant à des séries de termes ou de variables, et munies d'une interprétation fonctionnelle de façon similaire à la sémantique donnée plus loin pour les prédicats. N'en ayant pas l'usage dans la suite, nous excluons ces derniers éléments de cette présentation. En outre, toute fonction dans une signature peut être remplacée de façon équivalente par un prédicat d'arité supérieure dénotant une relation fonctionnelle.

Si une signature ne contient que des symboles du premier ordre, c'est-à-dire que $\rho_2(P) = 0$ pour tout P, la *logique du premier ordre* sur cette signature décrit l'ensemble de *formules* suivant :

— Si P est un prédicat, alors les expressions $P(x_1, \ldots, x_{\rho_1(P)})$ et $x = y$ (égalité) sont des formules dites *atomiques*.

— Si ϕ, ψ sont des formules, alors les expressions $\neg\phi$ (négation), $\phi \wedge \psi$ (conjonction), $\phi \vee \psi$ (disjonction), $\phi \Rightarrow \psi$ (implication), $\phi \Leftrightarrow \psi$ (équivalence), $\exists x.\phi$ (quantification existentielle) et $\forall x.\phi$ (quantification universelle) sont des formules dites *composites*.

L'ensemble des formules de la *logique monadique du second ordre* est composé de la même façon, en y ajoutant d'éventuels prédicats portant sur des variables du second ordre, ainsi que les constructions suivantes pour les formules :

— Les expressions $x \in X$ (appartenance) et $X \subseteq Y$ (inclusion) sont des formules atomiques.

— Si ϕ est une formule, les expressions $\exists^2 X.\phi$ et $\forall^2 X.\phi$ sont des formules composites.

Le qualificatif de « monadique » dénote le fait que les variables du second ordre peuvent être vues comme des prédicats unaires sur les variables du premier ordre. Ainsi, une formule atomique d'appartenance $x \in X$ dénote simplement l'application $X(x)$ du prédicat dénoté par X à la variable x. Plusieurs autres formules atomiques données ici peuvent être définies à partir d'un ensemble plus réduit de primitives, comme l'illustre la sémantique donnée plus bas ; cependant, nous les définissons comme des primitives par simplicité.

Comme pour les termes, une *sous-formule* est formule apparaissant dans une formule composite. Si une formule ϕ est une quantification, la variable qui lui est rattachée est dite *liée* dans toute sous-formule de ϕ. Toute variable

apparaissant dans une formule sans être liée par un quantificateur est dite *libre* dans cette formule ; et une formule *close* est une formule ne contenant pas de variables libres.

Sémantique, modèles Toute formule logique construite sur une signature dénote une affirmation portant sur une classe d'objets, dont le sens dépend de l'interprétation associée aux prédicats de la signature et aux variables libres de la formule. L'approche de la théorie des modèles confronte les formules logiques aux objets dont elles traitent (les modèles), en associant à toute formule une valeur de vérité dépendant d'un modèle et de la valuation des éventuelles variables libres de la formule dans un domaine fixé.

Plus précisément, pour une signature donnée, un *modèle* $\mathcal{M} = (\mathcal{D}, [\![])$ est formé par un *domaine* $\mathcal{D}$ et une *interprétation* $[\![]$ qui associe à tout prédicat P de la signature une fonction $[\![P]\!]$ de $\mathcal{D}^{\rho_1(P)} \times \mathscr{P}(\mathcal{D})^{\rho_2(P)}$ dans $\mathbb{B} = \{\mathit{Vrai}, \mathit{Faux}\}$. Enfin, une *valuation* est une fonction partielle de $\mathcal{X}_1$ dans $\mathcal{D}$ et de $\mathcal{X}_2$ dans $\mathscr{P}(\mathcal{D})$.

Étant donné un modèle $\mathcal{M}$, une valuation ν et une formule ϕ sur la signature du modèle, la *satisfaction* de ϕ par $\mathcal{M}$ et ν (notée $\mathcal{M}, \nu \models \phi$) est définie ainsi :

- $\mathcal{M}, \nu \models P\big(x_1 \ldots x_{\rho_1}, X_1 \ldots X_{\rho_2}\big)$ ssi $[\![P]\!]\big(\nu(x_1) \ldots \nu(x_{\rho_1}), \nu(X_1) \ldots \nu(X_{\rho_2})\big)$
- $\mathcal{M}, \nu \models x = y$ ssi $\nu(x) = \nu(y)$
- $\mathcal{M}, \nu \models x \in X$ ssi $\nu(x) \in \nu(X)$
- $\mathcal{M}, \nu \models \neg\phi$ ssi $\mathcal{M}, \nu \not\models \phi$ ($\mathcal{M}$ et ν ne satisfont pas ϕ)
- $\mathcal{M}, \nu \models \phi \wedge \psi$ ssi $\mathcal{M}, \nu \models \phi$ et $\mathcal{M}, \nu \models \psi$
- $\mathcal{M}, \nu \models \exists x.\phi$ ssi il existe $v \in \mathcal{D}$ tel que $\mathcal{M}, \nu' \models \phi$ si ν' est la fonction définie par $\nu'(y) = \nu(y)$ pour tout $y \in \mathrm{Dom}(\nu) \setminus \{x\}$ et $\nu'(x) = v$.
- $\mathcal{M}, \nu \models \exists^2 X.\phi$ ssi il existe $V \subseteq \mathcal{D}$ tel que $\mathcal{M}, \nu' \models \phi$ si ν' est la fonction définie par $\nu'(Y) = \nu(Y)$ pour tout $Y \in \mathrm{Dom}(\nu) \setminus \{X\}$ et $\nu'(X) = V$.

La notion de satisfaction pour les formules construites autrement peut être décomposée pour ne reposer que sur les primitives listées ci-dessus, de la manière suivante :

- $\mathcal{M}, \nu \models X \subseteq Y$ ssi $\mathcal{M}, \nu \models \forall x.x \in X \Rightarrow x \in Y$
- $\mathcal{M}, \nu \models \phi \Leftrightarrow \psi$ ssi $\mathcal{M}, \nu \models (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$
- $\mathcal{M}, \nu \models \phi \Rightarrow \psi$ ssi $\mathcal{M}, \nu \models \neg\phi \vee \psi$
- $\mathcal{M}, \nu \models \phi \vee \psi$ ssi $\mathcal{M}, \nu \models \neg(\neg\phi \wedge \neg\psi)$
- $\mathcal{M}, \nu \models \forall x.\phi$ ssi $\mathcal{M}, \nu \models \neg\exists x.\neg\phi$
- $\mathcal{M}, \nu \models \forall^2 X.\phi$ ssi $\mathcal{M}, \nu \models \neg\exists^2 X.\neg\phi$

La notion de satisfaction d'une formule en théorie des modèles donne lieu à la notion de langage d'une formule. Intuitivement, étant donné une signature, le langage $\mathcal{L}(\phi)$ d'une formule ϕ construite sur cette signature est l'ensemble des modèles pour lesquels il existe une valuation satisfaisant la formule ϕ. En particulier, le langage d'une formule close ϕ est $\mathcal{L}(\phi) = \{\mathcal{M} \mid \mathcal{M}, \emptyset \models \phi\}$, où $\emptyset$ désigne la valuation dont le domaine est l'ensemble vide. Par suite, un langage

L est dit *définissable* lorsqu'il existe une formule ϕ telle que $L = \mathcal{L}(\phi)$; cette notion trouvera son utilité dans le cadre des langages réguliers de termes, dans la section 2.3.4.

2.2 Lambda-calcul

Le lambda-calcul est un système formel, proposé par Church [1936b], qui permet de modéliser des expressions fonctionnelles et leur évaluation. Il permet en général de modéliser n'importe quelle fonction calculable, mais son expressivité peut être restreinte par l'ajout de contraintes supplémentaires. En particulier, l'ajout d'un système de typage dit simple permet de garantir la terminaison du calcul, tout en préservant une certaine capacité à manipuler des expressions.

2.2.1 Lambda-termes

Nous définissons tout d'abord les lambda-termes, qui représentent des expressions fonctionnelles.

Définition 2.1. Un *lambda-terme* est un terme construit inductivement à partir d'un ensemble de variables $\mathcal{X}$ (notées $x, y, z, \dots$) selon les règles suivantes :
- Si $x \in \mathcal{X}$, alors x est un lambda-terme (appelé *variable*).
- Si M est un lambda-terme et $x \in \mathcal{X}$, alors $(\lambda x.M)$ est un lambda-terme (appelé *abstraction* de x).
- Si M_1 et M_2 sont des lambda-termes, alors $(M_1 M_2)$ est un lambda-terme (appelé *application* de M_1 à M_2).

Par convention de notation, les parenthèses inutiles sont souvent omises selon les règles suivantes. L'application est prioritaire sur l'abstraction – c'est-à-dire que le terme $\lambda x.xy$ s'interprète implicitement comme $\lambda x.(xy)$, et non comme $(\lambda x.x)y$. De plus, l'application est associative à gauche – c'est-à-dire que xyz signifie $(xy)z$ et non $x(yz)$.

Les notions de variables libres et liées sont définies de manière similaire à celles de la logique formelle, les abstractions tenant dans cette définition le même rôle que les quantificateurs en logique.

Définition 2.2. Étant donné un lambda-terme M, l'ensemble des variables *libres* (resp. *liées*) de M est noté $\mathrm{FV}(M)$ (resp. $\mathrm{BV}(M)$) et défini comme suit :
- Si $M = x$, alors $\mathrm{FV}(M) = \{x\}$ et $\mathrm{BV}(M) = \emptyset$.
- Si $M = \lambda x.N$, alors $\mathrm{FV}(M) = \mathrm{FV}(N) \setminus \{x\}$ et $\mathrm{BV}(M) = \mathrm{BV}(N) \cup \{x\}$.
- Si $M = M_1 M_2$, alors $\mathrm{FV}(M) = \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2)$ et $\mathrm{BV}(M) = \mathrm{BV}(M_1) \cup \mathrm{BV}(M_2)$.

2.2.2 Sémantique opérationnelle

Les lambda-termes représentent des expressions fonctionnelles, où une abstraction dénote une définition de fonction, et une application représente l'application d'une fonction (à gauche) à son argument (à droite). Afin d'obtenir la sémantique recherchée pour l'évaluation d'une expression, le lambda-calcul définit plusieurs systèmes de réécriture sur les lambda-termes.

α-conversion

L'α-conversion est une relation qui dénote l'égalité entre deux expressions fonctionnelles modulo renommage de leurs paramètres – c'est à dire de leurs variables liées. Deux lambda-termes sont dits α-équivalents si l'un peut être obtenu à partir de l'autre par une série de renommages qui remplacent une variable dans la définition et dans le corps d'une fonction simultanément.

Définition 2.3. La relation de *α-renommage* entre deux lambda-termes M et M' (notée $M \to_\alpha M'$) est vérifiée dans les cas suivants :
- Si $M = \lambda x.N$ et que $M' = \lambda x.N'$ avec $N \to_\alpha N'$.
- Si $M = M_1 M_2$ et que $M' = M'_1 M_2$ avec $M_1 \to_\alpha M'_1$.
- Si $M = M_1 M_2$ et que $M' = M_1 M'_2$ avec $M_2 \to_\alpha M'_2$.
- Si $M = \lambda x.N$ et que $M' = \lambda y.N[x \leftarrow y]$ avec $y \notin \mathrm{FV}(N) \cup \mathrm{BV}(N)$.

Par extension, deux termes M et M' sont définis comme *α-équivalents* (noté $M =_\alpha M'$) si $M \overset{*}{\to}_\alpha M'$ ou si $M' \overset{*}{\to}_\alpha M$.

Les trois premiers cas dans la définition de $\to_\alpha$ stipulent que deux termes sont égaux par renommage s'ils peuvent être obtenus par renommage d'un sous-terme. Le dernier cas recouvre l'opération de renommage en elle-même, qui s'effectue directement par substitution ; afin de préserver le sens de l'expression fonctionnelle que dénote le terme renommé, cette substitution ne peut s'effectuer que si le nouveau nom de la variable (y) est absent du sous-terme où la substitution s'effectue.

Par la suite, nous adopterons la pratique usuelle consistant à considérer systématiquement des termes α-équivalents comme égaux, sans évoquer explicitement la notion d'α-conversion. Ce résultat peut être obtenu formellement en utilisant des indices de de Bruijn [1972] à la place des variables, qui dénotent la distance entre une variable et l'abstraction qui la lie : les termes ainsi obtenus sont indépendants de tout choix de dénomination de variables, et correspondent aux classes d'α-équivalence du lambda-calcul.

β-conversion

La β-contraction est une relation qui représente un « pas de calcul » dans l'évaluation d'une expression fonctionnelle. Le mécanisme de l'évaluation en lambda-calcul est la substitution d'une variable dans le corps d'une fonction

par l'argument auquel cette dernière est appliquée. Afin de préserver la sémantique des lambda-termes impliqués dans une évaluation, cette substitution doit également éviter la capture de variables. L'opération de β-contraction requiert donc une notion de substitution sans capture, s'appuyant au besoin sur un renommage des variables.

Définition 2.4. Une *substitution sans capture*, notée $M[N/x]$, est une substitution par N des occurrences libres de x dans M de sorte qu'aucune des variables libres de N n'est liée dans M. Cette opération peut impliquer une α-conversion implicite de M avant la substitution. Le terme $M[N/x]$ s'évalue donc inductivement comme suit :

— Si $M = x$, le résultat est N.
— Si $M = M_1 M_2$, le résultat est $M_1[N/x]\, M_2[N/x]$.
— Si $M = \lambda x.M'$, le résultat est M.
— Si $M = \lambda y.M'$ avec $y \neq x$ et $y \notin \mathrm{FV}(N)$, le résultat est $\lambda y.M'[N/x]$.
— Enfin, si $M = \lambda y.M'$ avec $y \neq x$ et que $y \in \mathrm{FV}(N)$, le résultat est $\lambda z.M'[y \leftarrow z][N/x]$; où z n'apparaît pas dans M ou dans N.

Le dernier cas de figure correspond au cas dans lequel le lambda-terme où a lieu la substitution requiert une α-conversion ; choisir une variable z « fraîche » résout dans ce cas les problèmes de capture éventuels.

Nous pouvons maintenant définir la β-contraction et ses relations dérivées.

Définition 2.5. Un *β-redex* est un lambda-terme formé d'une application dont la composante gauche est une abstraction.

La *β-contraction* relie un terme M contenant un β-redex $R = (\lambda x.P)N$ au terme M' obtenu par la substitution sans capture de x par N dans P. Formellement, la relation $M \to_\beta M'$ dénote que :

— Soit $M = (\lambda x.N)P$ et $M' = N[P/x]$.
— Soit $M = \lambda x.N$ et $M' = \lambda x.N'$ de sorte que $N \to_\beta N'$.
— Soit $M = M_1 M_2$ et $M' = M_1' M_2'$ de sorte que $M_1 = M_1' \wedge M_2 \to_\beta M_2'$ ou $M_1 \to_\beta M_1' \wedge M_2 = M_2'$.

La clôture réflexive/transitive $\xrightarrow{*}_\beta$ de $\to_\beta$ est nommée *β-réduction*, et la relation inverse *β-expansion*. La relation d'équivalence $=_\beta$ obtenue à partir de $\xrightarrow{*}_\beta$ est appelée *β-équivalence*.

Un lambda-terme ne contenant aucun β-redex est dit en *forme β-normale*.

Remarquons que les définitions de $\to_\alpha$ et $\to_\beta$ entraînent que le renommage des paramètres ne modifie pas le résultat de l'évaluation d'une fonction ; c'est à dire que si $M =_\alpha N$ et $M \to_\beta M'$, alors $N \to_\beta N'$ de sorte que $M' =_\alpha N'$.

De plus, la relation $\xrightarrow{*}_\beta$ présente une propriété importante, dite de Church-Rosser, qui garantit que tout choix entre deux redex dans l'opération de β-contraction n'est pas irréversible, en ce sens qu'il existe des réductions pour les termes résultants qui aboutissent au même terme.

Définition 2.6. Une relation $\rightarrow$ a la *propriété de Church-Rosser* si et seulement si $M \rightarrow M_1$ et $M \rightarrow M_2$ entraînent qu'il existe M' tel que $M_1 \rightarrow M'$ et $M_2 \rightarrow M'$.

Théorème 2.1. *La relation $\xrightarrow{*}_\beta$ a la propriété de Church-Rosser[cf. Barendregt, 1984, p. 59–62].*

Cette propriété entraîne immédiatement que si un lambda-terme M peut se réduire en N et que N est en forme β-normale, alors N est l'unique forme β-normale associée à M.

η-conversion

Enfin, la relation d'η-conversion dénote la vacuité d'une fonction f qui applique simplement une autre fonction f' à son argument, indépendamment de celui-ci. Elle permet d'éliminer dans ce cas l'abstraction et l'application inutiles, réduisant la fonction f à f'.

Définition 2.7. Un *η-redex* est un lambda-terme formé d'une abstraction de x sur l'application à x d'un terme où x n'est pas libre.

L'*η-contraction* relie un terme M contenant un η-redex $R = \lambda x.Nx$ au terme obtenu en substituant N à R dans M. Formellement, $M \rightarrow_\eta M'$ dénote :

- Soit $M = (\lambda x.M'x)$ et $x \notin \mathrm{FV}(M')$.
- Soit $M = \lambda x.N$ et $M' = \lambda x.N'$ de sorte que $N \rightarrow_\eta N'$.
- Soit $M = M_1 M_2$ et $M' = M'_1 M'_2$ de sorte que $M_1 = M'_1 \wedge M_2 \rightarrow_\eta M'_2$ ou $M_1 \rightarrow_\eta M'_1 \wedge M_2 = M'_2$.

Les notions d'*η-réduction* ($\xrightarrow{*}_\eta$), *η-équivalence* ($=_\eta$) et de *forme η-normale* sont définies de manière similaire à celles associées à la β-conversion.

En outre, les notions de *$\beta\eta$-contraction* ($\rightarrow_{\beta\eta}$), *$\beta\eta$-réduction* ($\xrightarrow{*}_{\beta\eta}$) et *$\beta\eta$-équivalence* ($=_{\beta\eta}$) sont définies pareillement pour le système produit à partir de la relation $M \rightarrow_{\beta\eta} M' \overset{def}{\Leftrightarrow} M \rightarrow_\beta M' \vee M \rightarrow_\eta M'$.

La relation $\xrightarrow{*}_\eta$ a trivialement la propriété de Church-Rosser, et cette propriété s'étend au système formé par $\xrightarrow{*}_{\beta\eta}$.

Théorème 2.2. *La relation $\xrightarrow{*}_{\beta\eta}$ a la propriété de Church-Rosser[cf. Barendregt, 1984, p. 65–67].*

2.2.3 Lambda-calcul simplement typé

Sans restriction, le lambda-calcul permet de modéliser le comportement de n'importe quelle fonction calculable. L'inconvénient majeur de cette expressivité est que l'existence d'un résultat final à l'évaluation d'un lambda-terme, c'est-à-dire d'une forme β-normale, n'est pas en général décidable. Ce fait est une conséquence immédiate de la principale propriété du problème de l'arrêt, qui stipule que le résultat de l'évaluation d'une fonction calculable est

indécidable en général[Turing, 1936][Church, 1936a]. Une restriction importante du lambda-calcul, qui permet de décider de l'équivalence entre deux termes, consiste à ne considérer que les termes dits simplement typés décrits par Church [1940], pour lesquels la terminaison du mécanisme d'évaluation est garantie. Nous décrivons d'abord deux méthodes classiques pour imposer cette restriction (des typages dits à la Curry et à la Church) et leurs conséquences, avant de préciser leur utilité dans la suite de la thèse.

Nous introduisons tout d'abord les notions de types simples et de signature typée.

Définition 2.8. Soit A un ensemble de symboles nommés *types atomiques*. L'ensemble $\mathcal{T}$ des *types simples* sur A (notés $\tau, \sigma, \dots$) est défini inductivement de la manière suivante : si $\tau \in A$, alors $\tau \in \mathcal{T}$; et si $\tau, \sigma \in \mathcal{T}$, alors $(\tau \to \sigma) \in \mathcal{T}$.

Une *signature* du lambda-calcul typé est formée par un ensemble $\mathcal{C}$ de *constantes* (notées par la suite $a, b, c, \dots$), et la définition d'un lambda-terme est étendue de sorte que toute constante est également un lambda-terme atomique. De plus, toute constante c est munie d'un type simple $\tau_c \in \mathcal{T}$.

Tout comme pour les lambda-termes, les parenthèses inutiles sont fréquemment omises dans les types simples, et l'opérateur $\to$ est associatif à droite : le type $\tau \to \tau \to \tau$ s'interprète comme $\tau \to (\tau \to \tau)$ et non comme $(\tau \to \tau) \to \tau$.

Le lambda-calcul simplement typé considère seulement des lambda-termes auxquels sont associés des types simples. Le constructeur $\to$ sur lequel sont basés ces derniers dénote l'abstraction fonctionnelle, en ce sens qu'une expression qui a pour type $\tau \to \sigma$ dénote une fonction dont l'argument est de type τ et dont le résultat de l'évaluation est de type σ. La notion de signature donnée ici répondra par la suite au même besoin que son pendant logique, où les constantes seront interprétées comme les éléments de base d'un modèle.

Typage à la Curry Nous introduisons maintenant un système de typage dit à la Curry, qui consiste à attribuer un type (par le biais d'un système d'inférence) à des lambda-termes. Celui-ci est basé sur les règles données par la figure 2.1.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\, Var \qquad \frac{c \in \mathcal{C}}{\Gamma \vdash c : \tau_c}\, Cst$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma}\, Abs \qquad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}\, App$$

FIGURE 2.1 – Système de typage simple pour le lambda-calcul

Définition 2.9. Le *système de typage* décrit figure 2.1 produit et manipule des *jugements* de typage de la forme $\Gamma \vdash M : \tau$, où τ est un type simple et M un

lambda-terme. Le membre gauche Γ, appelé un *environnement* de typage, est une fonction partielle, de domaine fini, de $\mathcal{X}$ dans $\mathcal{T}$. De façon transparente, la notation $\Gamma, x : \tau$ représente la fonction qui associe le type τ à x, et $\Gamma(y)$ à toute variable $y \neq x$; cette notation requiert implicitement que $x \notin \mathrm{Dom}(\Gamma)$.

Une inférence construite à partir de ces règles est appelée une *dérivation* de typage, et le jugement apparaissant en conclusion associe un type simple à un lambda-terme. S'il existe une dérivation dont la conclusion est de la forme $\Gamma \vdash M : \tau$, le lambda-terme M est dit (simplement) *typable*, et de type τ.

L'ensemble des lambda-termes typables est un sous-ensemble strict de celui des lambda-termes. Ces termes possèdent certaines propriétés immédiates en raison de leur définition : en particulier, tout sous-terme d'un terme typable est également typable. Une autre conséquence relativement immédiate est que les notions de α, β et η-conversion ainsi que leurs propriétés sont préservées sur les lambda-termes simplement typés. En outre, ceux-ci possèdent deux propriétés supplémentaires présentant un fort intérêt pour la suite.

Théorème 2.3 (Réduction du sujet)**.** *Si M est un lambda-terme typable de type τ et que $M \to_{\beta\eta} N$, alors N est également typable et de type τ [cf. Barendregt, 1993, p. 57].*

Théorème 2.4 (Normalisation forte)**.** *Le système de réécriture engendré par $\overset{*}{\to}_{\beta\eta}$ sur les lambda-termes simplement typés est fortement normalisant, c'est-à-dire qu'il ne contient pas de réduction infinie. Autrement dit, pour tout terme typable M, il existe $K \in \mathbb{N}$ tel que si $M \overset{k}{\to}_{\beta\eta} N$, alors $k \leq K$ [cf. Barendregt, 1993, p. 61].*

Notons que la conjonction de la propriété de normalisation forte et de la propriété de Church-Rosser entraînent que tout terme typable M possède une unique forme $\beta\eta$-normale, notée $|M|_{\beta\eta}$.

Typage à la Church Alternativement, le lambda-calcul simplement typé peut être obtenu par une méthode dite à la Church, en définissant directement l'ensemble des lambda-termes munis de leurs types. Les lambda-termes typés à la Church peuvent également être vus comme une notation pour une dérivation de type à la Curry portant sur un lambda-terme.

Définition 2.10. L'ensemble $\mathcal{X}$ des variables utilisées est complété par des annotations de type, construisant l'ensemble des variables typées $\mathcal{X} \times \mathcal{T}$. Ces variables sont notées avec leurs types en exposant : (x, τ) est ainsi noté x^τ. Une signature $\mathcal{C}$ est définie comme précédemment.

L'ensemble des lambda-termes typés est alors défini comme suit :
— Si $x^\tau \in \mathcal{X} \times \mathcal{T}$, alors x^τ est un lambda-terme de type τ (variable).
— Si $c \in \mathcal{C}$, alors c est un lambda-terme de type τ_c (constante).

— Si M est un lambda-terme de type σ et $x^\tau \in \mathcal{X} \times \mathcal{T}$, alors $\lambda x^\tau.M$ est un lambda-terme de type $\tau \to \sigma$ (abstraction).
— Si M_1 et M_2 sont des lambda-termes de type $\tau \to \sigma$ et τ respectivement, alors $M_1 M_2$ est un lambda-terme de type σ (application).

Par abus de notation, les annotations de type sur les variables sont parfois omises lorsqu'il n'y a pas d'ambigüité ; ainsi, le terme $\lambda x^\tau.x^\tau$ peut être noté $\lambda x^\tau.x$, les variables typées construites à partir d'une même variable étant alors considérées comme ayant le même type. Dans certains cas, le type d'une variable peut être laissé totalement implicite, par exemple lorsqu'il peut être déduit à partir du contexte fourni par une application (ainsi, y a implicitement le type τ dans l'expression $(\lambda x^\tau.x)\ y$).

Le lambda-calcul simplement typé ainsi obtenu possède – dans l'ensemble – la même sémantique que celle produite par un typage à la Curry, et il jouit des mêmes propriétés précédemment mentionnées (normalisation forte, réduction du sujet). En outre, comme il ne contient que des termes typables, il possède également la propriété d'*expansion du sujet* (la β-expansion préserve le type d'un terme). Le choix d'un système à la Church simplifie la question du typage en garantissant que tous les termes possèdent, par construction, un type immédiatement apparent ; mais n'offre pas, contrairement aux dérivations à la Curry, la possibilité de parler explicitement du processus permettant d'attribuer un type à un terme.

2.2.4 Représentations par le lambda-calcul typé

Choisir une signature appropriée et fixer une interprétation d'une manière similaire à la théorie des modèles permet d'utiliser des lambda-termes pour représenter différents objets d'intérêt. En particulier, il est possible de représenter tous les types d'objets définis dans la section 2.1 (mots, termes, contextes et formules logiques), ou des tuples de tels objets, par des lambda-termes sur une signature spécifique.

Ainsi, le monoïde libre sur un alphabet Σ peut être représenté en considérant l'unique type atomique $*$, et en construisant une signature composée de l'ensemble des constantes $\{a^{*\to *} \mid a \in \Sigma\}$ dénotant les lettres de Σ. Afin de représenter les termes de Σ^*, la concaténation est dénotée par le lambda-terme $. = \lambda s_1^{*\to *}.\lambda s_2^{*\to *}.\lambda x^*.s_1\ (s_2\ x)$. L'interprétation du lambda-calcul garantit alors l'associativité de la concaténation : $.\ f\ (.\ g\ h) =_\beta .\ (.\ f\ g)\ h =_\beta \lambda x^*.f\ (g\ (h\ x))$; de même que l'existence d'un élément neutre qui est l'identité $\varepsilon = \lambda x^*.x$. Tous les lambda-termes dénotant des mots ont ainsi le type $* \to *$.

Au delà de cette correspondance immédiate, la sémantique opérationnelle du lambda-calcul typé permet également de représenter des opérations sur ces mots, et de composer et d'évaluer le résultat ces dernières tout en garantissant leur terminaison. Il est ainsi possible de représenter des projections $\pi_i = \lambda s_1^{*\to *} \dots s_n^{*\to *}.s_i$, ou des tuples $\lambda\pi.\pi\ s_1 \dots s_n$ de taille n, ce qui permet

par exemple de dénoter des chaînes « à trous » (ou contextes), de manière transparente.

De manière similaire, il est possible de décrire des alphabets gradués, termes et contextes. Par convention de notation, τ^n est défini inductivement comme $\tau^{n-1} \to \tau$, avec $\tau^1 = \tau$ (par abus de notation, l'expression $\tau^0 \to \sigma$ désigne usuellement le type σ). Le type $*$ désigne alors un terme clos et $*^n \to *$ est le type d'un contexte d'arité n ; tout symbole s d'un alphabet gradué, d'arité $\rho(s) = n$, est alors représenté par une constante $s^{*^n \to *}$. La construction de termes est dénotée immédiatement par l'application des lambda-termes qui les représentent $(s\ t_1 \dots t_n)$, et les contextes s'obtiennent de manière similaire, avec une correspondance immédiate entre leurs variables et celles de leurs lambda-termes.

Cette construction s'étend aux formules logiques, en choisissant deux types e et t dénotant respectivement les termes ou variables du premier ordre, et les valeurs de vérité. Les variables du second ordre (et prédicats unaires) ont alors le type $e \to t$ et, plus généralement, un prédicat P d'arités $\rho_1(P) = n$, $\rho_2(P) = m$ est représenté par une constante P du type $e^n \to (e \to t)^m \to t$. Les connecteurs usuels sont également dénotés par des constantes, telles que : $\neg^{t \to t}$, $\wedge^{t \to t \to t}$, $\vee^{t \to t \to t}$, $\exists^{(e \to t) \to t}$ ou $\exists^{2((e \to t) \to t) \to t}$. La quantification d'une variable dans une formule est représentée en liant par une abstraction la variable en question dans le lambda-terme, puis en appliquant le terme résultant à la constante représentant le quantificateur, de la manière suivante : la formule $\exists x.P(x)$ est dénotée par le terme $\exists \lambda x^e.P\ x$. Intuitivement, un quantificateur peut être envisagé comme une opération qui, en fonction d'une formule dont la valeur de vérité dépend d'une variable donnée, « trouve » une valeur à substituer à cette variable afin de satisfaire la formule. Une conséquence pratique est que les notions de variables libres et liées coïncident entre une formule logique et le lambda-terme qui la représente.

2.2.5 Applications

Les grammaires catégorielles abstraites, décrites dans la section 2.4.2, utilisent des représentations par le lambda-calcul typé telles que celles évoquées ci-dessus pour décrire des langages d'objets sous la forme de langages de lambda-termes dénotant ces objets. Par suite, les linéarisations que nous décrivons dans le chapitre 3 s'appuient sur un lambda-calcul simplement typé à la Church. La notation des grammaires commutatives du chapitre 5 se base également sur ce principe ; et le système de réécriture qui permet de minimiser les termes et contextes impliqués dans nos dérivations repose sur un système de typage sémantique à la Curry 5.6 disposant prouvablement de certaines des propriétés décrites plus haut.

2.3 Les langages réguliers de termes

Nous étudions maintenant la notion de langages de termes, et en particulier la classe des langages réguliers de termes. Celle-ci possède plusieurs caractérisations simples et de bonnes propriétés de clôture ; en outre, nous verrons que l'image de ses langages par certaines classes de transductions produit des classes d'objets reconnaissables et pertinentes pour la formalisation linguistique.

Nous présentons ci-dessous plusieurs caractérisations des langages réguliers de termes, en précisant la façon dont elles sont reliées. Nous listons également plusieurs des propriétés de clôture de ces langages, et détaillons en particulier la connexion entre logique et automates qui est centrale tout au long du chapitre 3. Enfin, nous préciserons l'intérêt particulier que représentent les langages de termes dans le reste de la thèse.

2.3.1 Automates de termes

La première caractérisation que nous considérons est celle donnée par les automates à états finis. Ceux-ci permettent de décider aisément de l'appartenance d'un terme à un langage régulier, et d'établir simplement les différentes propriétés de clôture des langages réguliers : complémentation, union et intersection, et image par morphisme linéaire et morphisme inverse.

Définition 2.11. Un *automate de termes* est un 4-uplet $\mathcal{A} = (\mathcal{S}, Q, Q_f, \Delta)$, où :

— $\mathcal{S}$ est un alphabet gradué.
— Q est un ensemble fini d'*états*.
— $Q_f \subseteq Q$ est un ensemble d'*états finaux*.
— Δ est une *relation de transition* définie comme suit.

Une relation de transition est décrite par un ensemble (fini) de *transitions*, qui sont des entrées de la forme $s\,(q_1 \dots q_n) \to q_0$, où $s \in \mathcal{S}$, $\rho(s) = n$, et $q_i \in Q$ pour tout i. L'automate $\mathcal{A}$ est dit *déterministe* si Δ est une fonction, et *complet* si cette fonction est totale.

Un automate $\mathcal{A}$ reconnaît un terme sur un alphabet $\mathcal{S}$ en assignant des états à ses feuilles puis à ses nœuds internes de bas en haut jusqu'à la racine, d'après sa fonction de transition Δ. Le langage régulier associé à $\mathcal{A}$ est formé de l'ensemble des termes tels que $\mathcal{A}$ assigne à la racine du terme un état qui est final.

Définition 2.12. Un *parcours* d'un automate $\mathcal{A} = (\mathcal{S}, Q, Q_f, \Delta)$ sur un terme t est une fonction $\rho : \mathrm{Dom}(t) \mapsto Q$ qui respecte la fonction de transition Δ, en ce sens que si $t|_p = s(t_1 \dots t_n)$ et que $\rho(pi) = q_i$ pour tout $i \in [n]$, alors $s\,(q_1 \dots q_n) \to \rho(p)$ appartient à Δ.

Un parcours est dit *acceptant* si et seulement si $\rho(\varepsilon) \in Q_f$. Un automate $\mathcal{A}$ *reconnaît* un terme t si et seulement si il existe un parcours acceptant de $\mathcal{A}$ sur t. Enfin, le langage $\mathcal{L}(\mathcal{A})$ d'un automate $\mathcal{A}$ est l'ensemble des termes reconnus par $\mathcal{A}$.

Nous supposons dans la suite que tous les automates que nous manipulons sont déterministes et complets ; il est en effet possible pour tout automate $\mathcal{A}$ de produire un automate $\mathcal{A}'$ déterministe et complet tel que $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. Il est à noter que cette opération de déterminisation peut, dans le pire cas, augmenter exponentiellement le nombre d'états d'un automate.

La classe des langages dits réguliers, reconnus par les automates à états finis, possède plusieurs propriétés de clôture démontrables à partir de la définition des automates. Nous listons ici ces propriétés et une intuition de la construction de l'automate correspondant au langage résultant ; le détail et la preuve de ces constructions peuvent se trouver par exemple dans Comon *et al.* [2007]. Ces propriétés sont :

— Complémentation, en choisissant $Q'_f = Q \setminus Q_f$ comme nouvel ensemble d'états finaux sur un automate complet.
— Union, en construisant l'union disjointe des deux automates.
— Intersection, par conséquence directe des deux précédentes.
— Image par un morphisme linéaire, en appliquant le morphisme aux symboles dans les transitions de l'automate et en ajoutant là où c'est nécessaire des transitions pour reconnaître localement les contextes obtenus.

2.3.2 Grammaires de termes

Nous introduisons maintenant les grammaires régulières de termes, qui peuvent être vues comme des systèmes formels de réécriture générant des termes. Elles définissent des langages de termes constructibles (dérivables) selon les règles dont elles sont munies. Nous verrons que la classe de langages ainsi obtenue coïncide avec celle décrite par les automates à états finis.

Définition 2.13. Une *grammaire régulière de termes* G est un tuple $G = (\mathcal{S}, N, S, P)$, où :

— $\mathcal{S}$ est un alphabet gradué.
— N est un ensemble de *symboles non-terminaux*.
— $S \in N$ est un symbole non-terminal de départ, nommé *axiome*.
— P est un ensemble fini de *productions*.

Une production $p \in P$ est une paire $A \to t$ dont le membre gauche $A \in N$ est un symbole non-terminal, et le membre droit t est un terme construit sur l'alphabet $\mathcal{S} \cup N$, où les éléments de N sont d'arité nulle. Une production dont le membre droit ne contient pas de symboles non-terminaux est dite *terminale*.

Toute grammaire peut générer des termes par l'application répétée de ses productions : dans un terme construit sur $\mathcal{S} \cup N$ tout non-terminal A appa-

raissant à gauche d'une production de G peut être réécrit en le sous-terme apparaissant à droite de cette même production. En considérant pour terme initial l'axiome S de G, et par l'application de productions terminales (qui remplacent un symbole non-terminal par un terme sur $\mathcal{S}$), une grammaire permet de générer un ensemble de termes construits uniquement sur $\mathcal{S}$: cet ensemble constitue le langage de la grammaire.

Définition 2.14. Toute grammaire $G = (\mathcal{S}, N, S, P)$ induit une *relation de dérivation* $\rightarrow_G$ définie comme suit. Deux termes termes t et t' construits sur $\mathcal{S} \cup N$ sont reliés par $t \rightarrow_G t'$ si et seulement si il existe un contexte linéaire C tel que $t = C[A]$ et $t' = C[u]$, et que P contient la production $A \rightarrow u$.

Le langage $\mathcal{L}(A)$ d'un non-terminal A est constitué de l'ensemble des termes t tels que $A \overset{*}{\rightarrow}_G t$ et qu'aucun symbole non-terminal de N n'apparaît dans t. Le langage $\mathcal{L}(G)$ *généré* par la grammaire G est défini comme étant celui de son axiome, $\mathcal{L}(S)$.

Additionnellement, une grammaire telle que chacune de ses productions contient exactement un symbole de $\mathcal{S}$ est dite en *forme normale*. L'emploi de grammaires en forme normale simplifie certaines constructions et preuves, et n'entraîne pas de perte de généralité : il est toujours possible de décomposer une production dont le membre droit contient un terme complexe et d'éliminer les productions dont le membre droit se réduit à un symbole non-terminal, sans modifier le langage généré par la grammaire.

Correspondance grammaires/automates Nous établissons maintenant la correspondance entre grammaires et automates, qui est directe. Nous nous intéressons seulement à la traduction des grammaires vers les automates, afin de montrer que les langages générés par ces grammaires sont réguliers, la réciproque est toutefois vraie [*cf.* Comon *et al.*, 2007, p. 54]. Pour toute grammaire régulière de termes $G = (\mathcal{S}, N, S, P)$ en forme normale, il est possible de construire un automate de termes $\mathcal{A} = (\mathcal{S}, Q, Q_f, \Delta)$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(G)$ de la manière suivante :

- $Q = N$
- $Q_f = \{S\}$
- Si $A \rightarrow s(B_1 \dots B_n)$ est une production de P (où n peut être zéro), alors $s\,(B_1 \dots B_n) \rightarrow A$ est une transition de Δ, et Δ ne contient pas d'autre transition.

2.3.3 Définissabilité en logique

Comme mentionné dans la section 2.1.5, il est possible d'utiliser des formules logiques pour définir des langages par la théorie des modèles. Le langage d'une formule (généralement close) est alors l'ensemble des objets du domaine

qui satisfont cette formule, en suivant l'interprétation des prédicats appartenant à la signature sur laquelle la formule est construite. C'est précisément l'approche choisie par MTS, décrite dans la section 1.1.

Nous considérons maintenant des signatures dites à k successeurs, associées à un alphabet gradué $\mathcal{S}$ d'arité k, sur la logique monadique du second ordre : les langages des formules qui en résultent coïncident exactement avec la classe des langages réguliers de termes sur $\mathcal{S}$.

Définition 2.15. Étant donné un alphabet gradué $\mathcal{S}$ dont l'arité maximale est k, nous considérons la signature SkS formée des prédicats suivants, qui portent uniquement sur des variables du premier ordre :

— Si $s \in \mathcal{S}$, alors $s(x)$ est un prédicat d'arité 1 dans SkS.

— Pour tout $i \in [k]$, $S_i(x, y)$ est une relation d'arité 2 dans SkS.

La sémantique d'une logique construite sur SkS est obtenue en prenant pour classe de modèles l'ensemble des termes t construits sur $\mathcal{S}$, et pour domaine l'ensemble $\mathrm{Dom}(t)$ des positions valides dans t. Étant donné une valuation ν de $\mathcal{X}$ dans $\mathrm{Dom}(t)$, l'interprétation des prédicats de SkS est :

— $t, \nu \models s(x)$ ssi le sous-terme $t|_{\nu(x)}$ est construit par le symbole s.

— $t, \nu \models S_i(x, y)$ ssi $\nu(y) = \nu(x).i$.

La logique monadique du second ordre construite sur la signature SkS est appelée *logique monadique du second ordre à k successeurs* (abrégé en MSOkS).

La notion de langage d'une formule est définie exactement comme dans la section 2.1.5. Nous détaillons maintenant un résultat essentiel dû à Thatcher et Wright [1968], qui montre que les langages définissables par MSOkS coïncident exactement avec les langages réguliers de termes.

2.3.4 Correspondance logique/automates

Théorème 2.5. *Pour toute formule close ϕ de MSOkS, il existe un automate de termes $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$ [cf. Comon et al., 2007, p. 95].*

Une construction classiquement associée à cette preuve consiste à restreindre la syntaxe de MSOkS afin de transformer la formule ϕ en une formule équivalente ϕ' utilisant uniquement des variables du second ordre et un petit nombre de constructeurs, puis à construire l'automate $\mathcal{A}$ inductivement sur la structure syntaxique de ϕ'. Cette construction s'appuie essentiellement sur les propriétés de clôture des automates.

Syntaxe restreinte

Afin de construire la formule ϕ' à partir de ϕ, on s'appuie sur une syntaxe restreinte utilisant uniquement des variables du second ordre.

Afin d'obtenir des formules équivalentes à celles utilisant des variables du premier ordre, un nouveau prédicat $\mathrm{Sgl}(X)$ est introduit, qui dénote qu'une variable du second ordre est un singleton. Ce prédicat est défini comme suit :

$$\mathrm{Sgl}(X) \overset{def}{\Leftrightarrow} \neg(\forall^2 Z.X \subseteq Z) \wedge \forall^2 Y.Y \subseteq X \Rightarrow (X \subseteq Y \vee \forall^2 Z.Y \subseteq Z)$$

Cette définition garantit que l'ensemble dénoté par X est un singleton : X n'est pas vide, et tout sous-ensemble Y de X est soit X lui-même, soit l'ensemble vide.

Nous considérons maintenant la signature SkS, que nous altérons pour que ses prédicats portent sur des variables du second ordre. L'interprétation de la signature résultante est la suivante :

— $t, \nu \models s(X)$ ssi $t|_p = s(t_1 \dots t_n)$ pour tout p de $\nu(X)$.

— $t, \nu \models S_k(X, Y)$ ssi $p' = pk$ pour tout p de $\nu(X)$ et tout p' de $\nu(Y)$.

Observons que l'interprétation de la signature résultante coïncide avec celle de SkS sous l'hypothèse que les variables X et Y sont des singletons (contenant exactement les valeurs des variables du premier ordre x et y dans l'interprétation usuelle).

Nous pouvons maintenant, en nous appuyant sur la signature altérée et le prédicat $\mathrm{Sgl}(X)$, transformer inductivement toute formule close ϕ de MSOkS en une formule ϕ' équivalente en utilisant les règles suivantes :

— Si $\phi = s(x)$, alors $\phi' = s(X)$.

— Si $\phi = S_i(x, y)$, alors $\phi' = S_i(X, Y)$.

— Si $\phi = (x = y)$, alors $\phi' = (X = Y)$.

— Si $\phi = x \in Y$, alors $\phi' = X \subseteq Y$.

— Si $\phi = \neg\psi$, alors $\phi' = \neg\psi'$.

— Si $\phi = \psi_1 \wedge \psi_2$, alors $\phi' = \psi'_1 \wedge \psi'_2$.

— Si $\phi = \exists x.\psi$, alors $\phi' = \exists^2 X.\, \mathrm{Sgl}(X) \wedge \psi'$.

— Si $\phi = \exists^2 X.\psi$, alors $\phi' = \exists^2 X.\psi'$.

Intuitivement, cette transformation substitue à toute variable du premier ordre x une variable du second ordre X qui lui correspond, et garantit que cette variable dénote un singleton lorsqu'elle est quantifiée (par hypothèse, ϕ est une formule close). L'interprétation de ϕ est ainsi préservée, et les formules résultantes s'appuient uniquement sur des variables du second ordre, quantifications existentielles, conjonctions, négations et formules atomiques.

Construction inductive de $\mathcal{A}$

Nous construisons maintenant, pour toute formule ϕ utilisant la syntaxe restreinte, un automate $\mathcal{A}$ qui reconnaît un langage de termes associé à ϕ. Les termes sur lesquels travaille $\mathcal{A}$ représentent simultanément un modèle et une valuation, et ceux appartenant à $\mathcal{L}(\mathcal{A})$ sont exactement ceux correspondant à des paires modèle/valuation qui satisfont la formule ϕ.

Le résultat général est obtenu par induction sur une formule de départ close, en montrant initialement qu'à toute formule atomique correspond un automate. Il est ensuite possible d'associer inductivement à toute formule composite un automate construit en s'appuyant sur les propriétés de clôture données plus haut (complémentation, intersection, morphisme linéaire). Finalement, il suffit d'effacer des termes du langage résultant toute l'information portant sur la valuation qui satisfait la formule (sans conséquence puisque la formule d'origine est close).

Nous décrivons d'abord l'alphabet gradué $\mathcal{S}'$, sur lequel s'appuient les termes qui représentent une paire modèle/valuation. Soit $\mathcal{S}$ l'alphabet gradué sur lequel est basé SkS, et $\mathcal{X}_\phi \subset \mathcal{X}_2$ l'ensemble des variables susceptibles d'apparaître dans la formule ϕ. Remarquons que cet ensemble est fini (puisque la formule dont on souhaite construire le langage de termes est finie) et ne contient que des variables du second ordre ; on note son cardinal $\#(\mathcal{X}_\phi) = n$. L'alphabet $\mathcal{S}'$ est obtenu par le produit de $\mathcal{S}$ avec un tuple de booléens de taille n : $\mathcal{S}' = \mathbb{B}^n \times \mathcal{S}$; les arités de ses symboles sont données par leur composante droite : $\rho(b_1, \ldots, b_n, s) = \rho(s)$. Ainsi, la projection droite des symboles de $\mathcal{S}'$ dans un terme dénote un modèle $\mathcal{M}$ possible pour ϕ, et les autres composantes dénotent une valuation ν pour ses variables libres $X_1 \ldots X_n$ de la manière suivante : si le symbole d'un sous-terme à une position p a pour i^{e} composante *Vrai*, alors $p \in \nu(X_i)$. Cette interprétation est exemplifiée par la figure 2.2, où les valeurs booléennes *Faux* et *Vrai* sont représentées par 0 et 1.

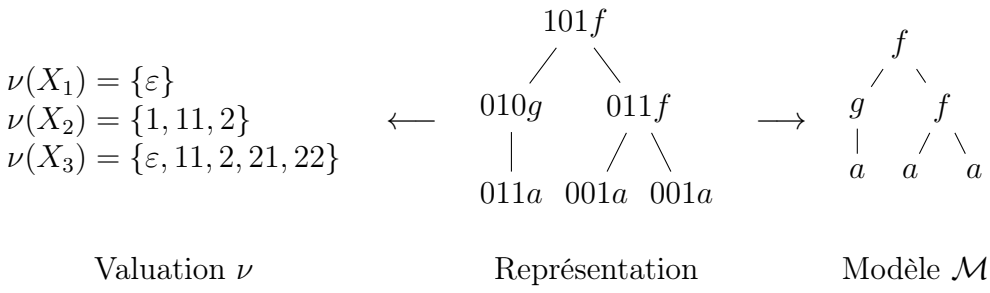


FIGURE 2.2 – Représentation d'un modèle et d'une valuation par un terme

Sans détailler leur construction explicite, nous donnons maintenant une indication des raisons pour lesquelles il existe un automate reconnaissant le langage L_ψ des termes qui dénotent un modèle et une valuation satisfaisant toute sous-formule atomique ψ de ϕ. Rappelons que la notation $\pi_i(s)$ dénote la i^{e} composante de s (c'est-à-dire dans ce cas l'appartenance de la position où est situé le symbole s à l'ensemble $\nu(X_i)$ dénoté par X_i) :

— Si $\psi = s(X_i)$, alors tout symbole s' présent dans un terme de L_ψ doit être tel que si $\pi_i(s') = \mathit{Vrai}$, alors leur dernière composante est s (toute position de $\nu(X_i)$ porte le symbole s). Cette propriété locale est aisément vérifiable par un automate simple.

— Si $\psi = S_l(X_i, X_j)$, considérons toute paire de positions (p, p') dans un terme de L_ψ dont les symboles sont (s_1, s_2) tels que $\pi_i(s_1)$ et $\pi_j(s_2)$ valent toutes les deux *Vrai* : alors cette paire de positions doit satisfaire $p' = p.l$ (p' est le l^{e} successeur de p). Cette propriété est également locale et peut être vérifiée par un automate de taille réduite, dont les états encodent la valeur de la j^{e} composante du dernier symbole parcouru.
— Si $\psi = (X_i = X_j)$, alors tout symbole s dans un terme de L_ψ doit vérifier $\pi_i(s) \Leftrightarrow \pi_j(s)$ (les valeurs des ensembles X_i et X_j coïncident).
— Si $\psi = X_i \subseteq X_j$, alors tout symbole s d'un terme de L_ψ doit vérifier $\pi_i(s) \Rightarrow \pi_j(s)$ (toute position de $\nu(X_i)$ appartient également à $\nu(X_j)$).

Nous traitons ensuite le cas des formules composites. Toute formule composite ψ de ϕ utilisant la syntaxe restreinte est soit une négation, soit une conjonction, soit une quantification existentielle du second ordre. Pour chacun de ces cas, le langage L_ψ des termes dénotant un modèle et une valuation satisfaisant ψ peut être décrit par un automate obtenu inductivement à partir des automates associées aux sous-formules immédiates de ψ pour les raisons suivantes :

— Si $\psi = \neg\psi'$, alors L_ψ est obtenu par complémentation de $L_{\psi'}$.
— Si $\psi = \psi_1 \wedge \psi_2$, alors L_ψ est l'intersection des langages L_{ψ_1} et L_{ψ_2}.
— Si $\psi = \exists^2 X_i.\psi'$, alors le langage L_ψ peut être obtenu en « oubliant » toute information sur l'ensemble X_i dans la valuation ν. Considérons le réétiquetage $h_i : (b_1 \dots b_{i-1}, b_i, b_{i+1} \dots b_n, s) \mapsto (b_1 \dots b_{i-1}, b_{i+1} \dots b_n, s)$: son application à un terme $t \in L_{\psi'}$ produit le terme $t' = h_i(t)$, dénotant la valuation attendue pour ψ (en supposant, sans perte de généralité, que la variable X_i n'apparaît dans aucune sous-formule de ϕ en dehors de ψ). Puisque les langages définis par les automates sont clos par morphisme linéaire (tel qu'un réétiquetage), le langage résultant est également régulier.

L'automate résultant reconnaît le langage L_ϕ des termes dénotant un modèle et une valuation satisfaisant ϕ ; puisque cette formule est close, la valuation ν dénotée par les n premières composantes est vide, ne laissant que l'information sur le modèle. L'automate $\mathcal{A}$ résultant reconnaît donc exactement l'ensemble des modèles qui, couplés à la valuation vide, satisfont la formule ϕ, c'est-à-dire que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi)$.

2.3.5 Application

Notre intérêt pour les langages de termes (et en particulier la classe des langages réguliers) est double. D'une part, les termes permettent d'encoder de manière transparente des arbres d'arité bornée, lesquels sont couramment utilisés pour représenter diverses structures du langage (structures syntaxiques, structures de traits, relations de dépendances, ...). D'autre part, des termes construits en tant qu'éléments d'une algèbre fixée peuvent être utilisés pour

représenter d'autres types d'objets (mots, graphes, ...), permettant de manipuler par des moyens algorithmiques simples toute structure représentable algébriquement.

La caractérisation donnée par les automates présentés initialement constitue pour nous une procédure de décision simple pour les langages réguliers, ainsi qu'un moyen d'établir simplement leurs propriétés de clôture. Ceci nous permet dans la suite de ramener des présentations alternatives, choisies pour leurs capacités descriptives, au cas des automates.

En revanche, le choix d'utiliser des grammaires de réécriture pour caractériser des langages réguliers de termes sera motivé dans le chapitre 3 par plusieurs avantages. Premièrement, les symboles non-terminaux et productions qu'elle contient donnent une structure visible au langage qu'elle génère : une production $p = A \to t(B_1 \dots B_n)$ s'interprète intuitivement comme la possibilité de construire un élément de type A par le biais d'un terme de la forme de t, où chaque B_i est remplacé par un sous-terme ayant le type B_i. Il est possible de donner une interprétation similaire aux transitions d'un automate, mais ceux-ci s'interprètent plus naturellement comme des outils de reconnaissance que comme des règles de génération.

Un autre intérêt, plus particulier à cette thèse, à l'emploi de grammaires régulières de termes, est leur similarité avec de nombreux mécanismes utilisés dans le cadre du traitement automatique des langues et de la formalisation linguistique, tels que les grammaires hors contexte de mots, les c-structures des grammaires lexicales-fonctionnelles ou les substitutions dans les grammaires d'arbres adjoints. Ces parallèles peuvent permettre à des usagers de ces formalismes n'ayant pas de notions approfondies en théorie des langages formels d'appréhender plus aisément les langages définis par des grammaires de réécritures.

Enfin, l'emploi de la logique formelle pour définir un langage de termes au moyen d'une formule sur MSOkS permet de décrire avec une grande concision, et indépendamment les unes des autres, les différentes propriétés des termes qui appartiennent à un langage – ou de ceux qui en sont spécifiquement exclus. Cette concision offerte par la logique, qui permet à des formules de taille réduite de décrire de très grands automates, peut être vue comme un obstacle algorithmique ; toutefois elle constitue également une source d'expressivité, qui nous permet de raccourcir et de simplifier grandement les descriptions linguistiques tout au long des chapitres 3 et 4.

De plus, la classe des langages réguliers de termes est incluse dans la classe des langages définissables par des grammaires de remplacement d'hyper-arêtes (HR), présentées dans Courcelle et Engelfriet [2011]. Cette dernière est close par une certaine classe de transductions dites MSO-définissables ; et les langages d'arbres qu'elle produit coïncident avec ceux définis par certaines classes de grammaires catégorielles abstraites (voir section 2.4.2). Cet état de fait nous permet d'établir quelques uns des résultats du chapitre 3, et constitue l'une

des motivations du travail de cette thèse.

Pour terminer, les grammaires commutatives présentées dans le chapitre 5 définissent des langages de termes dont l'interprétation produit indirectement les langages de mots étudiés. Les classes de langages de termes qui leurs sont associées incluent (mais ne se limitent pas à) la classe des langages réguliers.

2.4 Classes de langages et caractérisations

La linguistique informatique intègre dans ses hypothèses de travail le choix d'une classe de langages, conséquence des choix de conception du formalisme utilisé, qui constitue une approximation de la classe des langages naturels (ou du moins d'un sous-ensemble caractéristique de ceux-ci). Un exemple notable est la classe des langages légèrement sensibles au contexte, abordée dans la section 1.2.

À l'opposé des formalismes conçus selon des approches linguistiquement informées, d'autres caractérisations mathématiquement plus simples, issues de la théorie des langages, capturent des classes de langages similaires aux langages légèrement sensibles au contexte. Ces descriptions constituent à ce titre des candidats au titre d'outils « bas niveau » (à l'image des langages de programmation proches de la machine) pour la description de modèles linguistiques, et permettent d'étudier plus aisément les propriétés algorithmiques de ces classes de langages.

2.4.1 Grammaires hors-contexte multiples

Les grammaires hors-contexte multiples constituent une classe de grammaires décrivant des langages légèrement sensibles au contexte. Il existe plusieurs présentations équivalentes de ces grammaires[Seki *et al.*, 1991][Weir, 1988] ; nous adoptons ici celle donnée par Kanazawa [2010], où les productions peuvent s'interpréter directement d'une manière similaire aux clauses de Horn en programmation logique.

Définition 2.16. Une *grammaire hors-contexte multiple* (MCFG) G est définie comme un tuple $G = (\Sigma, N, S, P)$, dans lequel :

- Σ est un alphabet.
- N est un ensemble de symboles non-terminaux gradués.
- S est un élément de N, d'arité $\rho(S) = 1$ (l'axiome).
- P est un ensemble fini de productions de la forme :

$$A(u_1, \ldots, u_n) \leftarrow B_1(x_{1,1} \ldots x_{1,n_1}) \ldots B_p(x_{p,1} \ldots x_{p,n_p})$$

où, pour tout $k \in [n]$, u_k est un mot formé sur l'alphabet $\Sigma \cup \mathcal{X}$, l'ensemble $\mathcal{X} = \{x_{i,j} \mid i \in [p] \wedge j \in [n_i]\}$ dénotant l'ensemble des variables du membre droit.

Les productions de P doivent en outre respecter les conditions suivantes :
— Chaque variable de $\mathcal{X}$ apparaît au plus une fois dans le membre gauche.
— Pour tout $i \in [p]$, le symbole B_i est d'arité n_i, et A est d'arité n.

Une production dont le membre droit est vide est dite *terminale*. Une production dont le membre gauche contient toutes les variables de $\mathcal{X}$ est dite *non-effaçante*. Une MCFG est non-effaçante si toutes ses productions sont non-effaçantes.

Enfin, une m-MCFG est une MCFG dont tous les non-terminaux sont d'arité au plus m.

Une MCFG associe à chacun de ses non-terminaux un langage qui est un ensemble de tuples de mots. Ces tuples peuvent être dérivés successivement en interprétant intuitivement les productions comme des clauses de Horn, exprimant que le tuple $(u_1 \dots u_n)$ appartient au langage de A sous l'hypothèse que chaque $(x_{i,1} \dots x_{i,n_i})$ appartient au langage de B_i. Le langage de la grammaire s'obtient directement à partir de celui de son axiome.

Définition 2.17. Le langage $\mathcal{L}(A)$ associé par une grammaire G à un non-terminal A d'arité $\rho(A) = n$ est l'ensemble des tuples $(w_1 \dots w_n)$ respectant les conditions suivantes :
— $A(u_1, \dots, u_n) \leftarrow B_1(x_{1,1} \dots x_{1,n_1}) \dots B_p(x_{p,1} \dots x_{p,n_p})$ appartient à P.
— Pour tout $i \in [p]$, $(w_{i,1} \dots w_{i,n_i})$ appartient à $\mathcal{L}(B_i)$.
— Pour tout $k \in [n]$, $i \in [p]$ et $j \in [n_i]$, $w_k = u_k[x_{i,j} \leftarrow w_{i,j}]$.

Le langage $\mathcal{L}(G)$ d'une grammaire G est l'ensemble des mots contenus dans le langage de son axiome, $\mathcal{L}(S)$.

La caractérisation offerte par les MCFG permet de retrouver aisément d'autres classes de grammaires bien connues. Ainsi, les 1-MCFG équivalent aux grammaires hors-contexte (CFG) ; et restreindre les productions de P à la forme $A(w.x) \leftarrow B(x)$ (avec $w \in \Sigma^*$) produit la classe des grammaires régulières (RG). Une autre contrainte simple sur les productions des MCFG permet d'obtenir des grammaires dites sans entrelacement (*well-nested*, abrégées en MCFG_{WN}). Par suite, la classe de langages décrite par les 2-MCFG_{WN} coïncide avec les langages légèrement sensibles au contexte originellement envisagés par Joshi [1985], et les MCFG en général génèrent la même classe MCS (plus large) que les grammaires minimalistes et les les grammaires d'arbre adjoints multiples (*Multi-Component Tree Adjoining Grammars*).

Une dernière remarque sur le fonctionnement des MCFG est que, comme la plupart des grammaires génératives et des grammaires d'arbres, elles décrivent un ensemble d'arbres de dérivations formant un langage régulier (et même local) : ainsi, les choix de réécriture effectués pour un non-terminal donné n'affectent pas la réécriture des autres non-terminaux. Le tuple de mots généré par une dérivation peut par la suite être reconstruit en composant uniformément, de manière ascendante, les tuples associés aux sous-dérivations.

2.4.2 Grammaires catégorielles abstraites

Les grammaires catégorielles abstraites, proposées par de Groote [2001], offrent un cadre permettant de décrire de manière uniforme des langages d'objets représentables par le lambda-calcul simplement typé. Leur fonctionnement repose sur l'association entre un langage (dit abstrait) de structures grammaticales, et le langage (dit concret) des objets représentés par ces structures. Une grammaire catégorielle abstraite est décrite par son langage abstrait et par l'application (nommée lexique) associant les termes de ce langage à des termes de son langage concret.

Définition 2.18. Une grammaire catégorielle abstraite (ACG) G est définie comme un tuple $G = (\Sigma_a, \Sigma_c, \Lambda, s)$, où :

— Σ_a et Σ_c sont des signatures du lambda-calcul simplement typé (*cf.* définition 2.8), appelées respectivement le *vocabulaire abstrait* et le *vocabulaire concret* de G.

— Le *lexique* Λ de G est une fonction qui associe aux constantes de Σ_a des lambda-termes linéaires construits sur Σ_c. Cette fonction doit offrir un typage cohérent, en ce sens que tout type atomique de Σ_a doit correspondre à un unique type simple de Σ_c par Λ.

— Le *type distingué* s de G est un type simple de Σ_a.

L'ensemble des lambda-termes sur Σ_a ayant le type s est appelé le *langage abstrait* de G. L'image des termes du langage abstrait par le morphisme défini par Λ constitue le *langage concret* (ou *langage objet*).

Si les termes du langage concret d'une grammaire G peuvent être interprétés comme des objets mathématiques (mots, termes, graphes, ...), l'ensemble des objets ainsi dénotés peut être considéré de façon transparente comme le langage de G. À l'inverse, le langage abstrait peut être considéré, à l'image des arbres de dérivation d'une grammaire générative, comme l'ensemble des structures grammaticales « internes » de la grammaire G.

La classe des langages dénotés par l'ensemble des grammaires catégorielles abstraites est extrêmement riche. Pour cette raison, une restriction usuelle est de se limiter à un vocabulaire concret permettant de dénoter uniquement des mots sur un alphabet ou des termes d'une algèbre spécifique, en fixant une interprétation unique pour toutes les grammaires de ce type. Par suite, il est possible de décrire une hiérarchie sur les ACG résultantes, en restreignant la complexité du langage abstrait et du lexique qui le relie au langage concret.

Définition 2.19. L'*ordre* $\mathrm{ord}(\tau)$ d'un type simple τ est défini inductivement comme :

— $\mathrm{ord}(\tau) = 1$ si τ est un type atomique.

— $\mathrm{ord}(\sigma \to \tau) = \max(\mathrm{ord}(\sigma) + 1, \mathrm{ord}(\tau))$ autrement.

Par suite, l'*ordre* $\mathrm{ord}(\Sigma)$ d'un vocabulaire Σ est défini comme l'ordre maximal des types des constantes de Σ ; et la *complexité* $\mathrm{comp}(\mathscr{L})$ d'un lexique

$\mathscr{L}$ est définie comme l'ordre maximal des images des types atomiques de Σ_a (cette valeur étant fixée en raison de la cohérence du typage induit par $\mathscr{L}$).

La notation $\mathbf{ACG}(n, m)$ désigne l'ensemble des ACG dont le vocabulaire abstrait est d'ordre n et dont le lexique est de complexité m. Par extension, une grammaire $G \in \mathbf{ACG}(n, m)$ est dite d'ordre n et de complexité m.

Cette notation induit une double hiérarchie sur l'ensemble des grammaires catégorielles abstraites. Ainsi, la classe de langages induite par les ACG du premier ordre correspond aux langages finis. Les grammaires du second ordre induisent, en considérant une représentation canonique des mots par des lambda-termes telle que celle donnée dans la section 2.2.4, des classes de langages connues, selon la complexité de leur lexique :

Complexité	1	2	3	4+
Classe équivalente	RG	CFG	MCFG_{WN}	MCFG

Des résultats similaires existent pour les langages d'arbres représentables par des ACGKanazawa [2009a] (allant des langages réguliers pour $m = 1$, aux langages représentables par des grammaires de remplacement d'hyper-arêtes pour $m \geq 4$). Un autre résultat remarquable apparaît en considérant la classe des langages de mots induite à partir d'une classe de langages d'arbres par l'opération de concaténation de gauche à droite des feuilles des arbres (usuellement nommée *yield*) : l'image par cette opération de la classe des langages d'arbres induite par les $\mathbf{ACG}(2, m)$ est la classe des langages de mots induite par les $\mathbf{ACG}(2, m + 1)$.

2.4.3 Applications

La hiérarchie des MCFG est présentée ici non seulement pour des raisons illustratives, mais surtout parce qu'elle sert de support à la hiérarchie des grammaires commutatives construite et étudiée dans le chapitre 5, qui propose une manière d'étendre la notion de sensibilité légère au contexte aux langues non-configurationnelles. L'encodage permettant d'obtenir la correspondance entre les ACG du second ordre et les MCFG, dû à de Groote et Pogodalla [2004], est essentiellement identique à celui qui est donné dans la section 5.3.2.

Enfin, le formalisme de description linguistique décrit dans le prochain chapitre s'inspire directement des ACG dans sa structure générale ; construisant un langage abstrait de structures grammaticales, il permet ensuite de décrire des linéarisations de ces structures vers plusieurs langages concrets, dont le rôle est similaire à celui du lexique dans une ACG. Bien que ces linéarisations (et le langage abstrait qu'elles transforment) soient essentiellement décrits de manière implicite par la logique, nous mettrons en correspondance les langages de lambda-termes qui en résultent avec ceux générés par les ACG du second ordre.

Chapitre 3

Définition du formalisme

L'objectif initial de cette thèse est la mise au point d'un formalisme comprenant d'une part un puissant langage de description linguistique, et capable d'autre part de produire une caractérisation formelle des langages résultants, permettant l'analyse ou la génération automatique d'énoncés. En particulier, le but est de pouvoir fournir à un linguiste un outil de description simple, permettant d'implémenter directement des phénomènes linguistiques (notamment sous la forme de contraintes de grammaticalité) ; et d'en obtenir automatiquement un analyseur syntaxique, capable – si possible – de travailler en un temps polynomial relativement à la taille de l'énoncé analysé. C'est la poursuite de ce double objectif qui a guidé les décisions de conception et le travail présentés dans ce chapitre.

À l'image des grammaires transformationnelles ou des ACG, le formalisme résultant construit d'abord la structure profonde (ou abstraite) d'un énoncé, et en déduit ensuite ses représentations phonologiques ou sémantiques (les structures concrètes). Il décrit dans un premier temps l'ensemble des structures abstraites comme des arbres, représentant de façon hiérarchique les composants syntaxiques de l'énoncé. La correction syntaxique (au sens linguistique) de ces structures est établie par un ensemble de contraintes de bonne formation. Celles-ci sont modélisées par des formules logiques, une structure abstraite étant considérée comme bien formée lorsqu'elle vérifie l'ensemble des formules logiques qui lui sont rattachées. Cette approche est similaire à celle de MTS[Rogers, 1996], hormis que les modèles acceptés représentent seulement la structure abstraite des phrases correctes, et non leur forme phonologique. En particulier, les questions liées à l'ordre des mots sont exclues de cette description, et sont traitées dans un second temps.

L'autre versant du formalisme est la description de linéarisations, c'est-à-dire de fonctions transformant une structure abstraite en sa représentation concrète. Cette dernière peut être une chaîne de caractères représentant la forme phonologique de l'énoncé, mais également une représentation sémantique (à la manière de Montague [1974]). Le terme de « linéarisation » sera

employé dans tous les cas par abus de langage. Tout comme les conditions de grammaticalité des énoncés, la relation entre structures abstraites et concrètes s'appuiera sur des contraintes logiques. L'emploi de plusieurs linéarisations associant à un même ensemble de structures abstraites plusieurs représentations concrètes peut alors permettre d'effectuer des tâches liées au traitement automatique des langues, comme l'extraction d'informations, la génération de texte ou la traduction automatique. Comme dans les ACG, la structure abstraite sert alors d'étape intermédiaire pour passer d'une représentation concrète à une autre.

Tout au long de ce chapitre, nous illustrerons notre méthodologie de description linguistique au travers d'un langage artificiel basique, formé d'expressions conditionnelles (si/alors/sinon) et arithmétiques (opérations binaires), qui servira de support pour montrer le fonctionnement du formalisme. L'application de celui-ci à la modélisation de phénomènes linguistiques réels fera ensuite l'objet d'un traitement à part, tout au long du chapitre 4.

3.1 Structures abstraites

Nos *structures abstraites* sont des arbres, dont les nœuds représentent les composants syntaxiques de l'énoncé, et les feuilles les éléments du lexique. Les arêtes sont étiquetées par des fonctions syntaxiques, dénotant la relation entre le sous-arbre qu'elles dominent et son nœud parent. Par convention, toutes les arêtes sortantes d'un même nœud portent des étiquettes différentes. Un inconvénient de ce choix est que les adjonctions deoivent se faire de manière récursive, plutôt que de permettre, par exemple, à de multiples adverbes de s'appliquer au même niveau. Cela facilite en revanche la visualisation de la forme générale des structures abstraites dans la grammaire d'approximation.

En raison de cette convention, et comme l'ensemble des étiquettes d'arêtes apparaissant dans une grammaire est fixé, tous les arbres d'une grammaire sont de degré borné ; ce qui permet de les considérer comme des termes. Tous les nœuds internes de ces termes sont construits au moyen d'un même symbole (noté $\bullet$) dont l'arité est égale au nombre d'étiquettes d'arête utilisées dans la grammaire. Par conséquent, chaque étiquette d'arête correspond à un entier positif inférieur ou égal à l'arité de $\bullet$, et les sous-termes correspondant à des sous-arbres inexistants (c'est-à-dire dont la fonction syntaxique n'est pas remplie) sont remplacés par un symbole $\bot$, d'arité nulle. Enfin, les éléments du lexique sont représentés par autant de symboles, également d'arité nulle.

La figure 3.1 illustre cette correspondance, en présentant à gauche une structure abstraite sous forme d'arbre, et à droite le terme qui l'encode. Cette structure abstraite appartient à notre langage d'exemple, et correspond à l'expression « si vrai alors 1 + 1, sinon 0 ». L'ensemble des étiquettes d'arête susceptibles d'apparaître est alors : $\{si, alors, sinon, op, arg_1, arg_2\}$; le symbole

• des nœuds internes est donc d'arité 6 dans le terme à droite. L'ensemble des feuilles utilisées (les éléments du lexique) est décrit dans la prochaine section.

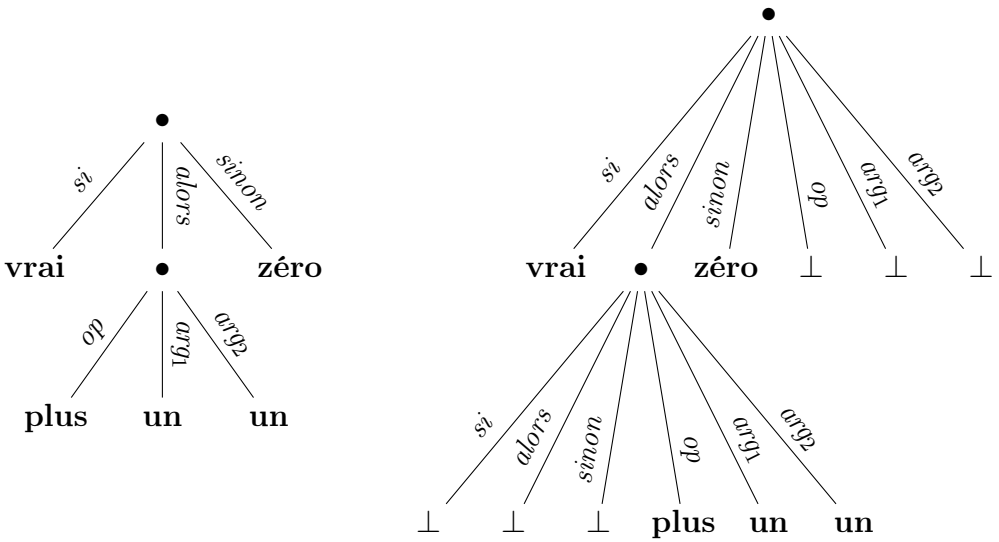


FIGURE 3.1 – Représentation et implémentation d'une structure abstraite

Dans la suite, nous emploierons systématiquement des arbres pour représenter les structures abstraites, en omettant les feuilles étiquetées par ⊥ et leurs arêtes entrantes des représentations, sauf pour expliciter des détails d'implémentation (nos résultats techniques s'appuyant sur l'encodage sous forme de termes).

Du point de vue des modélisations linguistiques à venir, nos choix de représentation quant à la syntaxe profonde des phrases nous conduisent à utiliser une étiquette d'arête spécifique pour représenter la tête d'un syntagme (c'est-à-dire sa projection lexicale) et, comme ici, à ne pas distinguer les nœuds internes de la structure (tous construits à l'aide du même symbole •). Une alternative aurait été de placer la tête syntagmatique directement sur les nœuds internes, à la manière de Tesnière [1959] (comme illustré par le déplacement de l'opérateur **plus** dans la figure 3.2) ; cela aurait toutefois compliqué la modélisation de l'adjonction, en forçant tous les adjoints à être rattachés au même niveau, et ainsi à utiliser un encodage vers les termes moins transparent. Une autre possibilité envisagée initialement était d'étiqueter les nœuds internes par des catégories syntaxiques, mais cet étiquetage est apparu comme artificiel et, par la suite, redondant avec l'usage des non-terminaux dans les grammaires d'approximation décrites plus bas.

3.1.1 Entrées lexicales

L'ensemble des étiquettes utilisées pour les feuilles est répertorié dans un *lexique*, dont les entrées associent chaque étiquette à un ensemble de traits,

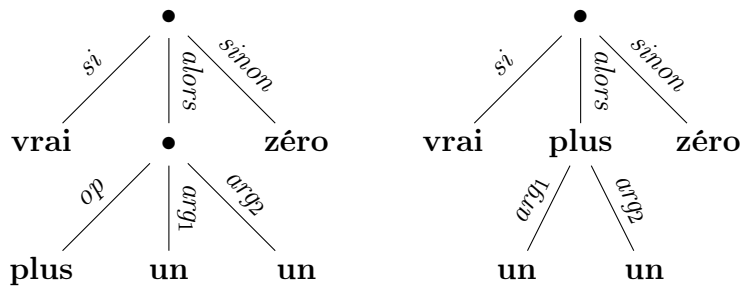


FIGURE 3.2 – Exemple de placement de l'information sur les nœuds internes

nommés *propriétés* de l'entrée lexicale. Ces propriétés seront utilisées pour contraindre la grammaticalité des structures abstraites : elles incluront dans le prochain chapitre (consacré aux descriptions linguistiques) des informations syntaxiques telles que les parties du discours, marques d'accord, sous-catégorisation, restrictions de sélection, *etc.*

Par hypothèse, un lexique contient un nombre fini d'entrées et de propriétés associées à chaque entrée. Un lexique minimal pour notre langage d'exemple est donné par la table 3.1, incluant des propriétés permettant de distinguer les types de valeurs (booléen, entier), les opérateurs et leur type (logique, arithmétique), ainsi que l'arité (unaire ou non) et les priorités (faible ou forte) de ces derniers.

Entrée	Propriétés
vrai	booléen
faux	booléen
zéro	entier
un	entier
ou	opérateur ; logique ; priorité −
et	opérateur ; logique ; priorité +
non	opérateur ; logique ; priorité + ; unaire
plus	opérateur ; arithmétique ; priorité −
moins	opérateur ; arithmétique ; priorité −
fois	opérateur ; arithmétique ; priorité +

TABLE 3.1 – Exemple de lexique

Nous donnons maintenant une définition formelle d'un lexique, construit à partir d'un ensemble fini de mots et d'un ensemble fini de propriétés.

Définition 3.1. Étant donné un ensemble fini de symboles L appelés *mots du lexique* et un ensemble fini de symboles P appelés *propriétés* (lexicales), une *entrée lexicale* est une paire (l, P_l), où $l \in L$ et $P_l \in \mathscr{P}(P)$.

Un *lexique* $\mathscr{L}$ est simplement défini comme un ensemble (fini) d'entrées lexicales.

3.2 Grammaire d'approximation

Afin de caractériser l'ensemble des structures abstraites valides – c'est-à-dire qui dénotent des énoncés syntaxiquement corrects – nous nous appuierons dans un premier temps sur une grammaire régulière de termes. Les termes qu'elle décrit correspondent à des structures abstraites arborescentes, selon l'encodage décrit précédemment. Cette grammaire vise uniquement à spécifier la forme générale des structures abstraites, sans modéliser toutes les règles de bonne formation des énoncés. Le langage ainsi généré illustre simplement la structure récursive de la langue, et est une grossière sur-approximation de l'ensemble des structures abstraites bien formées ; il sera affiné par la suite par l'ajout de contraintes de bonne formation. Dans la suite, nous désignerons cette grammaire indifféremment sous le terme de *grammaire d'approximation* ou de *grammaire support* (en raison des contraintes et des règles de linéarisation qui viendront décorer ses productions par la suite). Cette approche hybride n'est pas sans rappeler celle de Boral et Schmitz [2012], qui filtre les arbres de dérivation d'une grammaire hors-contexte à l'aide de la logique dynamique propositionnelle sur les arbres.

Les non-terminaux d'une grammaire d'approximation dénotent le type des structures abstraites qu'ils génèrent. En outre, plutôt que d'énumérer toutes les règles lexicales possibles, nous emploierons par souci de concision des propriétés issues du lexique en guise de symboles terminaux : implicitement, ces symboles dénotent n'importe quelle entrée du lexique possédant la propriété correspondante.

La figure 3.3 fournit une grammaire d'approximation possible pour notre langage d'exemple en listant ses productions. Elle possède un unique symbole non-terminal E, désignant une expression (arithmétique ou logique). La première production (en haut à gauche) permet de construire des expressions conditionnelles (« si [condition] alors [expression 1], sinon [expression 2] »), la production suivante (à droite) permettant de combiner une ou deux expression(s) au moyen d'un opérateur (la notation « (E) » est explicitée à la fin de cette section), et les deux productions terminales (en bas) permettent de réécrire une expression comme une valeur entière ou booléenne respectivement.

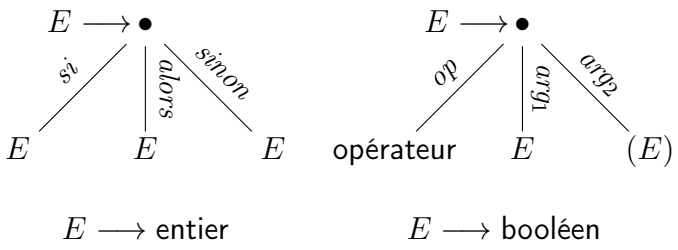


FIGURE 3.3 – Exemple de grammaire d'approximation

Observons que cette grammaire n'opère pas de distinction entre les expres-

sions et opérateurs logiques ou arithmétiques, ce qui permet de produire, entre autres, des expressions que l'on souhaiterait considérer comme mal formées, comme illustré par la figure 3.4. Celle-ci comporte deux structures abstraites appartenant au langage décrit par la grammaire d'approximation ; la seconde étant dénuée de sens, puisqu'elle combine deux nombres entiers au moyen d'un opérateur logique unaire portant sur des booléens. Ces structures seront par la suite filtrées au moyen des contraintes de bonne formation évoquées précédemment.

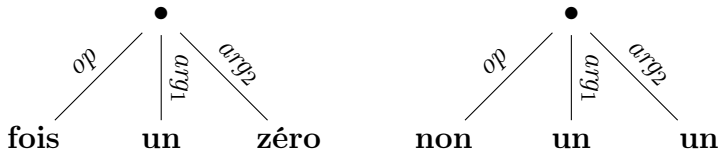


FIGURE 3.4 – Exemples de structures abstraites non-contraintes

Les productions d'une grammaire d'approximation seront toujours représentées avec leur membre droit sous sa forme arborescente pour faciliter la lecture. Pour des raisons pratiques, nous emploierons également un raccourci de notation dans les membres droits des productions : certains non-terminaux seront notés entre parenthèses. Cette écriture dénote l'optionnalité : le non-terminal correspondant peut soit être réécrit de la façon usuelle, soit être simplement absent de la structure résultante. Alternativement, la notation (A) peut être interprétée comme un non-terminal distinct de A, auquel sont implicitement associées les deux productions $(A) \to A$ et $(A) \to \bot$.

Dans le cadre du formalisme décrit ici, l'emploi d'une grammaire d'approximation pour spécifier la forme générale des structures abstraites n'est pas indispensable : il est possible d'obtenir le même ensemble de structures en utilisant directement des contraintes logiques (qui décrivent également la classe des langages réguliers de termes). Cependant, l'emploi d'une grammaire permet de disposer d'un « squelette », qui servira par la suite de support pour imposer des contraintes sur certaines réécritures et pour décrire plus aisément la linéarisation des structures abstraites vers des formes concrètes. En particulier, la lisibilité des contraintes logiques décrites par la suite bénéficie grandement de l'ancrage visuel fourni par les productions de la grammaire.

3.3 Contraintes logiques

Afin de contraindre la grammaticalité des structures abstraites, nous allons ensuite ajouter à la grammaire support des règles de bonne formation exprimées par des formules logiques. L'ensemble des arbres du langage abstrait sera alors restreint à l'ensemble des termes générés par la grammaire d'approximation qui satisfont également toutes les contraintes logiques associées. Le choix

d'un langage logique pour exprimer ces contraintes a d'importantes conséquences : il détermine l'expressivité du formalisme, ainsi que la complexité algorithmique des problèmes de décision associés. Dans notre cas, la logique monadique du second ordre sur une signature à k successeurs (MSOkS) s'impose comme un candidat (maximal) naturel : comme mentionné précédemment (voir section 2.3.4), l'ensemble des modèles satisfaisant à ses formules constitue un langage régulier, là où la satisfiabilité pour les logiques d'ordre supérieur n'est pas généralement décidable. En outre, il ressort de notre travail de modélisation décrit dans le chapitre 4 que de nombreuses règles de correction syntaxique issues de plusieurs langues sont exprimables au moyen de ce langage, ce qui suggère que cet outil permet de modéliser la syntaxe des langues en général. Nos formules seront donc construites en utilisant un sous-ensemble de MSOkS.

Nous allons d'abord détailler la nature exacte de ce vocabulaire logique, puis nous décrirons la façon dont les contraintes logiques enrichissent la grammaire d'approximation.

3.3.1 Vocabulaire logique

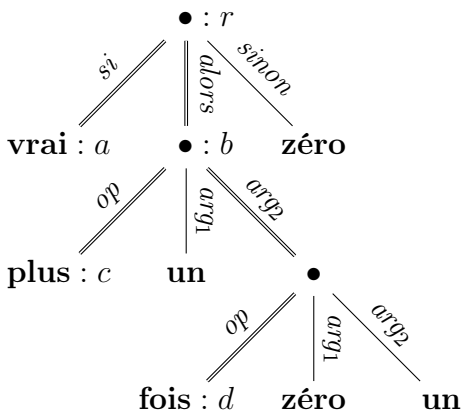
À l'usage, il apparaît que toute l'expressivité de la logique monadique du second ordre sur les termes n'est pas requise pour exprimer des contraintes sur la structure des phrases à un niveau abstrait. En particulier, la capacité de quantifier sur des ensembles de positions ne semble pas à première vue avoir d'application immédiate en linguistique. Toutefois, la restriction à une simple logique du premier ordre sur les termes ne permet pas d'exprimer de manière adéquate les phénomènes dits de dépendance à distance, tels que les contraintes d'îlot modélisées dans le chapitre suivant (voir section 4.4). Ces derniers requièrent d'une façon ou d'une autre la capacité d'exprimer des relations portant sur des chemin de longueur non-bornée entre deux positions – ce qui est exprimable en utilisant MSOkS. Aucun des autres phénomènes rencontrés jusqu'à présent n'ayant requis plus d'expressivité, nos contraintes logiques seront construites en utilisant la logique du premier ordre sur une signature à k successeurs, enrichie des ensembles de relations suivants :

— Pour toute expression régulière r construite sur l'ensemble des étiquettes d'arête, $r(x, y)$ est une relation binaire qui est vraie si et seulement si il existe un mot w dans le langage de r tel que que le chemin de x à y est étiqueté par w.

— Pour toute paire d'expressions régulières r_1, r_2 sur l'ensemble des étiquettes d'arête, $r_1 \uparrow r_2(x, y)$ est une relation qui est vraie si et seulement si le plus petit ancêtre commun z de x et y est tel que le chemin de z à x est étiqueté par un mot de r_1 et celui de z à y par un mot de r_2.

La figure 3.5 illustre le fonctionnement de ces relations. Elle représente une structure abstraite dans laquelle certaines positions sont étiquetées par

des variables (a, b, c, d et r pour la racine). Considérons l'expression régulière $(arg_1 \mid arg_2)^*\ op$: la relation $(arg_1 \mid arg_2)^*\ op\,(x, y)$ qu'elle définit est vérifiée pour toute paire de positions x, y telle que x domine y par une série de zéro ou plus arêtes étiquetées par arg_1 ou arg_2, suivies d'une arête étiquetée op. Dans la figure, cette relation est vérifiée pour les paires b, c et b, d, dont les chemins portent respectivement les séries d'étiquettes op et $arg_2\ op$.



$$si \uparrow alors\ (arg_1 \mid arg_2)^*\ op$$

FIGURE 3.5 – Structure abstraite et relations sur des chemins réguliers

Par suite, la relation plus complexe $si \uparrow alors\ (arg_1 \mid arg_2)^*\ op$ donnée par la figure est vérifiée pour les paires a, c et a, d : dans chaque cas, le plus petit ancêtre commun des deux nœuds (en l'occurrence la racine r) domine a par une arête étiquetée si et c ou d par un chemin étiqueté $alors\ op$ ou $alors\ arg_2\ op$ respectivement, satisfaisant les expressions régulières données de part et d'autre du symbole $\uparrow$; les chemins correspondants étant ceux marqués par une double barre dans la figure.

La signature MSOkS employée inclut en outre k relations de succession, qui correspondent aux k étiquettes d'arête utilisées dans la grammaire. Aussi, par confort de lecture dans nos contraintes logiques, nous dénoterons la relation S_i entre deux positions en utilisant l'étiquette d'arête associée à l'entier $i \in [k]$ dans les termes représentant les structures abstraites, en suivant la correspondance décrite plus haut (section 3.1, §2). Par exemple, nous écrirons $sinon(x, y)$ au lieu de $S_3(x, y)$, en suivant la correspondance suggérée par la figure 3.1.

Cette signature comprend également les prédicats unaires liés aux symboles de l'alphabet gradué ; lesquels incluent, dans notre cas, le symbole $\bullet$ d'arité k, la feuille $\perp$, et les feuilles associées aux entrées du lexique. Par la suite, les nœuds internes $\bullet$ ne véhiculant pas d'information, le prédicat $\bullet(x)$ ne sera pratiquement pas utilisé. En revanche, afin de se rapporter aisément aux pro-

priétés du lexique, nous ajouterons un ensemble de prédicats nommés d'après ces propriétés ; ceux-ci seront vérifiés pour toute position étiquetée par une entrée lexicale possédant la propriété correspondante. Ainsi, dans la figure 3.5, la position a satisfait le prédicat $\mathsf{booléen}(a)$, tandis que la position c satisfait les prédicats $\mathsf{opérateur}(c)$ et $\mathsf{arithmétique}(c)$.

Cette logique servira de support à la fois pour définir notre notion de grammaticalité et pour décrire les linéarisations vers des structures concrètes. Elle est notre principal outil de description linguistique et, à cet effet, doit pouvoir exprimer de manière concise n'importe quel concept linguistique à l'œuvre dans nos structures abstraites. Aussi, nous souhaitons éviter d'avoir à construire ou répéter des formules logiques complexes – en particulier lorsqu'elles dénotent un concept linguistique qui n'est pas immédiatement apparent dans nos structures abstraites. Pour cela, nous enrichirons notre vocabulaire logique au fur et à mesure de la conception d'une grammaire, au moyen de l'opérateur $\triangleq$. Classiquement, le prédicat introduit à gauche de l'opérateur sera par la suite interprété comme une abréviation pour la formule logique à droite. Ce mécanisme s'avérera très utile pour construire de nouvelles primitives linguistiques itérativement, en s'appuyant sur celles définies jusqu'alors. Le principal avantage de cette approche est que les formules logiques construites à partir de ces alias ont une bien meilleure lisibilité que celles bâties simplement sur la signature logique de départ, ce qui contribue à la facilité de compréhension et à la maintenabilité des grammaires.

Définition formelle Nous allons maintenant définir formellement la signature logique sur les termes utilisée par la suite, ainsi que les prédicats supplémentaires associés. Soit $\mathscr{L}$ le lexique utilisé, P l'ensemble de ses propriétés, et E l'ensemble des étiquettes d'arêtes. Pour commencer, nous rappelons la construction habituelle de l'ensemble des expressions régulières sur E.

Définition 3.2. L'ensemble $\mathcal{R}(E)$ des *expressions régulières sur* E est défini comme suit :

- $\emptyset$ est une expression régulière, qui dénote le langage vide : $\mathcal{L}(\emptyset) = \emptyset$.
- ε est une expression régulière, qui dénote le mot vide : $\mathcal{L}(\varepsilon) = \{\varepsilon\}$.
- Si $e \in E$, e est une expression régulière, qui dénote le langage $\mathcal{L}(e) = \{e\}$.
- Si $r_1, r_2 \in \mathcal{R}(E)$, $r_1 \mid r_2$ est une expression régulière qui dénote l'union des langages : $\mathcal{L}(r_1 \mid r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$.
- Si $r_1, r_2 \in \mathcal{R}(E)$, $r_1 r_2$ est une expression régulière, qui dénote la concaténation des langages : $\mathcal{L}(r_1 r_2) = \{w_1 w_2 \mid w_1 \in \mathcal{L}(r_1) \wedge w_2 \in \mathcal{L}(r_2)\}$.
- Si $r \in \mathcal{R}(E)$, r^* est une expression régulière, qui dénote l'itération du langage : $\mathcal{L}(r^*) = \{w^n \mid w \in \mathcal{L}(r) \wedge n \in \mathbb{N}\}$.

Nous décrivons maintenant l'ensemble des formules logiques sur la signature qui produit nos structures abstraites.

Définition 3.3. Étant donné un ensemble dénombrable de variables $\mathcal{X}$ notées $x, y, z, \ldots$:

— Si $l \in \mathscr{L}$, $l(x)$, $\bullet(x)$ et $\perp(x)$ sont des formules atomiques.
— Si $r \in \mathcal{R}(E)$, $r_1(x, y)$ est une formule atomique.
— Si $r_1, r_2 \in \mathcal{R}(E)$, $r_1 \uparrow r_2(x, y)$ est une formule atomique.
— Si ϕ et ψ sont des formules, $\neg\phi$, $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \Rightarrow \psi$, $\phi \Leftrightarrow \psi$, $\exists x.\phi$ et $\forall x.\phi$ sont des formules.

Par souci de lisibilité, le prédicat $\perp(x)$ (correspondant à l'absence d'une entrée lexicale) sera dénoté par la formule sans(x), et sa négation sera abrégée par la formule avec(x). De plus, l'expression régulière formée par la disjonction de toutes les étiquettes d'arêtes présentes dans $E = \{e_1 \ldots e_n\}$ sera abrégée en dom $= e_1|\ldots|e_n$, afin de pouvoir exprimer simplement la dominance immédiate (ou au sens large) entre deux nœuds – en écrivant par exemple dom(x, y) ou dom$^*(x, y)$.

À l'exception des expressions régulières, l'ensemble des formules obtenues est un sous-ensemble strict de MSOkS n'utilisant pas de variables du second ordre ; la sémantique qui leur est associée est également la même que celle indiquée dans la section 2.1.5. Les formules atomiques $r(x, y)$ dénotent pour leur part que x domine y de telle sorte que les étiquettes des arêtes sur le chemin de x à y forment un mot appartenant à $\mathcal{L}(r)$. Cette interprétation peut-être obtenue en transformant les expressions régulières en une formule équivalente de la logique monadique du second ordre. Pour toute expression régulière $r \in \mathcal{R}(E)$, son interprétation $[\![r]\!]$ peut être obtenue inductivement comme suit :

— $[\![\emptyset(x, y)]\!] \overset{def}{\Leftrightarrow} Faux$
— $[\![\varepsilon(x, y)]\!] \overset{def}{\Leftrightarrow} x = y$
— $[\![e(x, y)]\!] \overset{def}{\Leftrightarrow} S_i(x, y)$
— $[\![r_1 \mid r_2(x, y)]\!] \overset{def}{\Leftrightarrow} [\![r_1(x, y)]\!] \vee [\![r_2(x, y)]\!]$
— $[\![r_1 r_2(x, y)]\!] \overset{def}{\Leftrightarrow} \exists z.\, [\![r_1(x, z)]\!] \wedge [\![r_2(z, y)]\!]$
— $[\![r^*(x, y)]\!] \overset{def}{\Leftrightarrow} \exists^2 Z.x \in Z \wedge \forall z.z \in Z \Rightarrow (\exists z' \neq z.z' \in Z \wedge [\![r(z, z')]\!]) \vee z = y$

Comme évoqué précédemment, les étiquettes d'arête $e(x, y)$ dénotent les relations de succession $S_i(x, y)$ (avec $i \in [k]$) correspondantes dans la définition usuelle de MSOkS – rappelons que l'égalité $\#(E) = ar(\bullet)$ permet d'établir cette correspondance.

La formule qui traduit l'itération utilise un quantificateur du second ordre pour construire un ensemble Z de positions intermédiaires entre x et y : cet ensemble inclut x, et tout nœud z qu'il contient doit soit dominer un autre nœud z', également inclus dans Z, par un chemin reconnu par r (effectuant un pas d'itération) ; soit être le nœud y lui-même (le « dernier » élément de Z).

Enfin, les formules atomiques construites à partir de l'opérateur $\uparrow$ décrivent un chemin montant dans l'arbre depuis x jusqu'à un plus petit ancêtre commun, puis redescendant vers y. Elles peuvent donc être interprétées de la manière suivante (la dernière clause traduisant le fait que z est le plus petit ancêtre

commun à x et y) :

$$[\![r_1 \uparrow r_2(x,y)]\!] \overset{def}{\Leftrightarrow} \exists z.[\![r_1(z,x)]\!] \wedge [\![r_2(z,y)]\!] \\ \wedge (\forall z'.\, [\![\mathrm{dom}^*(z',x)]\!] \wedge [\![\mathrm{dom}^*(z',y)]\!] \Rightarrow [\![\mathrm{dom}^*(z',z)]\!])$$

3.3.2 Contraintes de grammaticalité

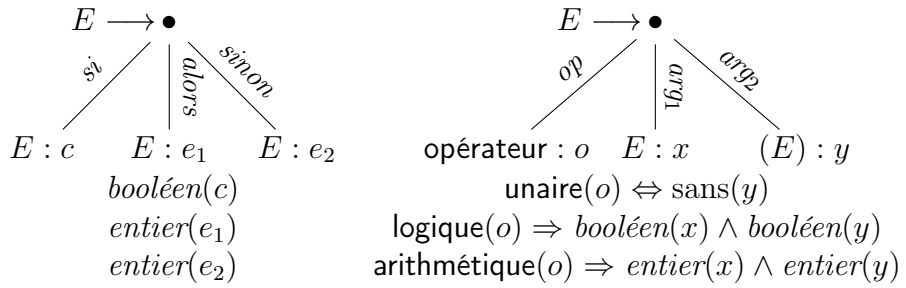
Maintenant que nous disposons d'un vocabulaire logique nous permettant de formuler des *contraintes logiques* de grammaticalité, nous allons décrire la manière dont ces contraintes vont enrichir nos grammaires d'approximation.

Nous décorons les productions de ces dernières par deux moyens : d'une part, les nœuds internes et feuilles du membre droit d'une production peuvent être étiquetés par des variables de $\mathcal{X}$; d'autre part, chaque production est associée à un ensemble de formules logiques, lesquelles peuvent contenir comme variables libres les étiquettes apparaissant dans le membre droit de leur production. Une grammaire ainsi décorée par des contraintes logiques est appelée une *grammaire enrichie*.

Reprenant notre exemple précédent, nous enrichissons la grammaire d'approximation permettant de construire des expressions au moyen de trois règles pour chacune des deux premières productions, comme illustré dans la figure 3.6. Les contraintes ajoutées s'appuient sur deux prédicats supplémentaires *booléen* et *entier*, définis au bas de la figure : une valeur v (correspondant à une position arbitraire dans l'arbre) satisfait le prédicat *booléen* lorsqu'elle correspond à une entrée lexicale possédant la propriété booléen, lorsqu'elle domine un opérateur logique (par le biais d'une arête étiquetée *op*), dont le résultat est, par hypothèse, une valeur booléenne, ou lorsqu'elle se réduit à $\perp$. Le prédicat *entier* est défini de manière similaire, en ajoutant un cas supplémentaire pour inclure le résultat d'une expression conditionnelle (par hypothèse un entier).

Par suite, les contraintes sur les expressions conditionnelles (formées par la première production) imposent que leurs arguments soient respectivement une valeur booléenne (c) et deux valeurs entières (e_1 et e_2). Les contraintes sur la seconde production imposent quant à elles, respectivement : que l'absence de l'argument y, optionnel, coïncide avec l'emploi d'un opérateur unaire, que l'emploi d'un opérateur logique suppose des arguments booléens, et de même qu'un opérateur arithmétique reçoit des arguments entiers. Le même langage abstrait aurait pu être obtenu en multipliant les symboles non-terminaux : en employant par exemple E_a et E_l comme symboles non-terminaux pour des expressions arithmétiques et logiques respectivement, et E^u ou E^b pour des expressions employant des opérateurs unaires ou binaires, et en spécifiant chacune des productions correspondantes. L'emploi de contraintes logiques vise à éliminer ces énumérations lorsqu'aucune interaction n'entre en jeu entre plusieurs concepts, comme ici entre l'arité et le type des opérateurs.

Lorsqu'une structure abstraite appartient au langage décrit par une grammaire d'approximation, il existe une dérivation de cette structure qui associe



$$
\begin{aligned}
\textit{booléen}(v) \;&\triangleq\; \mathsf{booléen}(v)\\
&\vee\; \exists o.\, op(v,o) \wedge \mathsf{logique}(o)\\
&\vee\; \mathrm{sans}(v)\\
\textit{entier}(v) \;&\triangleq\; \mathsf{entier}(v)\\
&\vee\; \exists o.\, op(v,o) \wedge \mathsf{arithmétique}(o)\\
&\vee\; \exists c.\, si(v,c) \wedge \mathrm{avec}(c)\\
&\vee\; \mathrm{sans}(v)
\end{aligned}
$$

FIGURE 3.6 – Exemple de grammaire enrichie

des productions de la grammaire à des nœuds internes de la structure : une production est associée à un nœud lorsque celui-ci a été obtenu par la réécriture d'un non-terminal qui est permise par cette production. Cette association permet également de faire correspondre une instance d'une variable décorant le membre droit d'une production à une position dans la structure abstraite résultante. Une dérivation d'une structure abstraite est alors dite *valide* si et seulement si toutes les contraintes logiques associées aux productions utilisées dans cette dérivation sont satisfaites. Cette notion de validité d'une dérivation s'étend naturellement aux structures abstraites : une structure abstraite est valide si et seulement si il existe une dérivation valide de cette structure d'après la grammaire enrichie considérée.

La figure 3.7 illustre la notion de validité à travers un exemple de structure abstraite, en donnant une dérivation complète de celle-ci conforme à la grammaire d'approximation donnée plus haut, et en listant en bas les contraintes qu'elle doit vérifier. Les arêtes en pointillé de l'arbre de dérivation représentent les choix de réécriture, en indiquant les productions utilisées : les productions de la grammaire d'approximation sont numérotées de p_1 à p_4, un $\mathscr{L}$ indique une règle lexicale (réécriture d'une propriété par une entrée lexicale correspondante) et la mention « opt. » indique l'omission d'un non-terminal optionnel. Les productions p_1 et p_2, utilisées une fois chacune, instancient leurs contraintes logiques aux positions correspondantes dans la structure abstraite (remarquons que la position 13, non représentée dans la structure abstraite, correspond à une feuille $\perp$.). Chacune de ces contraintes étant vérifiée dans la structure abs-

traite en vertu des définitions des prédicats *booléen* et *entier*, la structure est considérée comme valide.

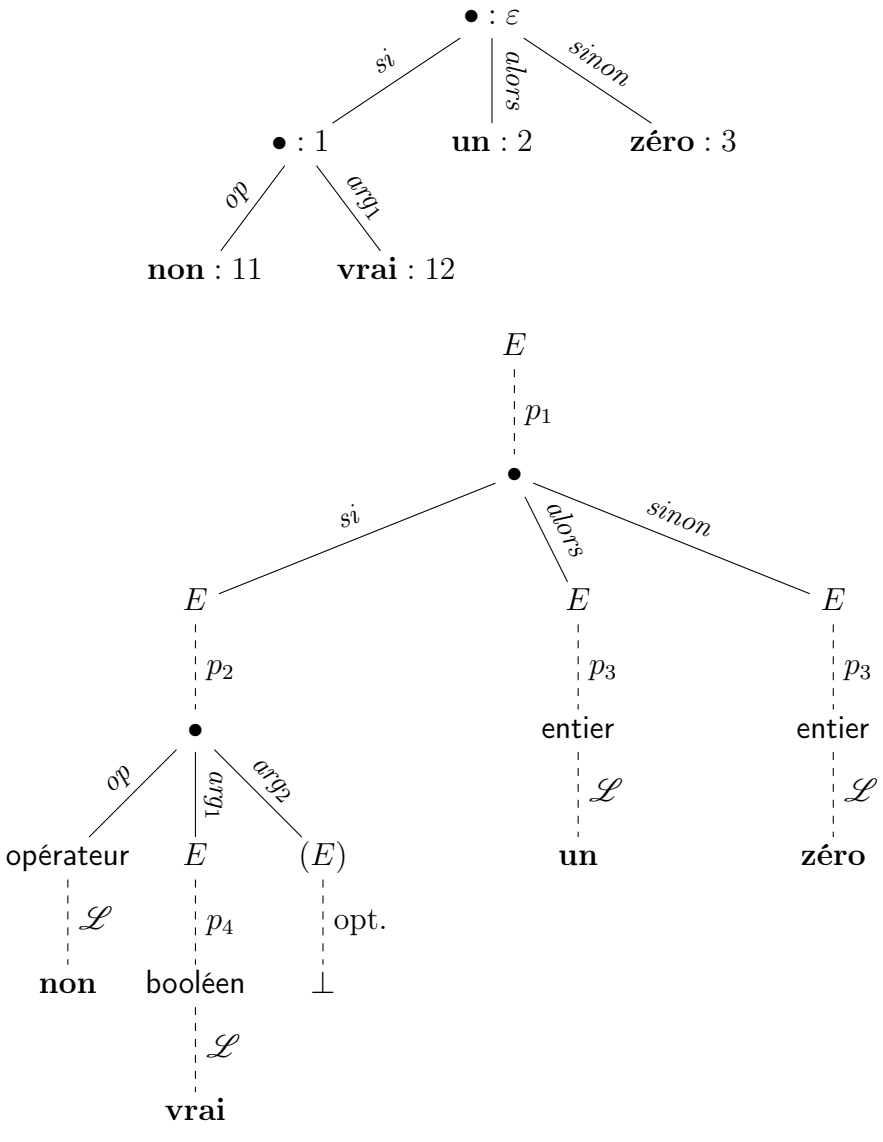
En pratique, ces contraintes logiques nous permettront de restreindre le langage abstrait en modélisant des contraintes linguistiques sur l'optionnalité de certains arguments, la sous-catégorisation, les restrictions de sélection et, le cas échéant, en imposant des dépendances à longue distance entre deux composants syntaxiques. En raison de la correspondance entre logique et automates décrite dans le chapitre 2, les grammaires enrichies par des contraintes ont le même pouvoir d'expression que les grammaires d'approximation dans l'absolu : la différence principale réside dans la concision qu'elles offrent pour décrire des langages. La prochaine sous-section illustre ce résultat en compilant une grammaire enrichie vers un langage régulier de termes.

Ce choix de compléter par des contraintes une grammaire d'approximation simple pour définir un langage abstrait est motivé par la volonté de ne pas surcharger la grammaire principale, et de conserver sa lisibilité. La construction d'une grammaire de réécriture modélisant plusieurs phénomènes du langage entraîne en effet rapidement, comme suggéré plus haut, une multiplication combinatoire des non-terminaux et des productions, qui doivent transmettre diverses informations sur le contexte de réécriture. Par contraste, nos contraintes logiques ajoutent aux productions de la grammaire une forme limitée (finie) de sensibilité au contexte. L'emploi de contraintes logiques permet ainsi de simplifier la construction de grammaires de grande taille, répondant aux mêmes besoins que la tradition des méta-grammaires évoquée section 1.3.

3.3.3 Compilation du langage abstrait

Une grammaire enrichie, telle que celle proposée ci-dessus, décrit un ensemble de structures abstraites valides, c'est-à-dire un langage de termes. Celui-ci est composé de l'ensemble des termes dérivables par la grammaire d'approximation qui satisfont l'ensemble des contraintes instanciées par leur dérivation. Ce langage de termes est en outre régulier, en raison de la connexion entre logique et automates décrite précédemment (voir section 2.3.4) ; nous en donnons maintenant une caractérisation effective, construite à partir d'une grammaire enrichie G.

La première étape est de convertir G afin que toutes ses contraintes logiques portent sur les racines de ses productions, puis d'interpréter les non-terminaux optionnels et règles lexicales. Par suite, l'idée centrale est de construire une grammaire régulière de termes G' (sans contraintes logiques) où les contraintes apparaissent directement dans les termes du langage, en construisant un alphabet gradué dont les symboles sont associés aux formules logiques de la grammaire. L'ensemble des contraintes de la grammaire G étant fini, nous pouvons alors construire une formule MSOkS qui vérifie que tout nœud étiqueté par une contrainte satisfait cette contrainte. Le langage résultant est



p_1	p_2
booléen(1)	unaire(11) $\Leftrightarrow$ sans(13)
entier(2)	logique(11) $\Rightarrow$ *booléen*(12) $\wedge$ *booléen*(13)
entier(3)	arithmétique(11) $\Rightarrow$ *entier*(12) $\wedge$ *entier*(13)

FIGURE 3.7 – Dérivation et validation d'une structure abstraite

régulier, et son intersection avec le langage sans contraintes de G' (également régulier) garantit que les formules ne sont instanciées que là où les production de G le permettent. Rappelons que les propriétés des langages réguliers de termes citées dans la section 2.3.1 incluent leur clôture par intersection et morphisme linéaire : le langage résultant de l'intersection précédente est donc régulier. De plus, par construction, il ne diffère du langage associé à G que par les symboles de son alphabet gradué (qui portent des formules logiques en plus des symboles utilisés dans G) ; or, un réétiquetage effaçant les formules pour ne conserver que l'information d'origine constitue un type simple de morphisme linéaire, ce qui achève la construction du langage régulier associé à G.

Nous détaillons maintenant cette construction. Formellement, la grammaire enrichie G que nous devons compiler est définie par son lexique, l'ensemble de ses productions enrichies et l'ensemble de ses symboles non-terminaux, incluant son symbole de départ. Son lexique $\mathscr{L}$ est construit sur un ensemble fini de mots du lexique L et un ensemble fini de propriétés P (voir définition 3.1). Soit $\mathcal{N}$ l'ensemble de ses symboles non-terminaux, et S son symbole de départ. Nous définissons maintenant précisément la nature de nos productions enrichies.

Définition 3.4. Soit $\mathcal{S} = \{\bullet, \bot\} \cup \{A, (A) \mid A \in \mathcal{N}\} \cup P$ l'alphabet gradué utilisé dans les productions enrichies, incluant l'ensemble des symboles non-terminaux, y compris optionnels (ceux apparaissant entre parenthèses), ainsi que toute propriété de P issue du lexique (employée comme symbole terminal) ; tous les symboles sont d'arité nulle, à l'exception de $\bullet$ (d'arité k). L'ensemble des variables pouvant être utilisées dans les contraintes est noté $\mathcal{X}$.

Une *production enrichie* est un objet de la forme $A \to t; \Lambda; \Phi$, où :
- $A \in \mathcal{N}$ est un symbole non-terminal, nommé *membre gauche*.
- t est un terme construit sur l'alphabet gradué $\mathcal{S}$, nommé *membre droit*.
- $\Lambda : \mathrm{Dom}(t) \mapsto \mathcal{X}$ est une fonction partielle dite *d'étiquetage* de t.
- Φ est une *contrainte*, c'est-à-dire une conjonction de formules logiques respectant la définition 3.3.

Une production enrichie est *bien formée* lorsque sa fonction d'étiquetage Λ est injective – afin d'interdire à une même variable d'étiqueter deux positions distinctes du membre droit. L'ensemble des productions enrichies d'une grammaire G est noté $\mathcal{R}$.

Le processus de compilation de G esquissé plus haut se décompose ainsi :

1. Interprétation des non-terminaux optionnels.
2. Ajout des règles lexicales.
3. Déplacement des contraintes logiques vers la racine des membre droits.
4. Construction de la grammaire régulière de termes G'.
5. Construction d'une formule MSOkS Val vérifiant les contraintes.
6. Construction du langage de G à partir de $\mathcal{L}(G')$ et de $\mathcal{L}(\mathrm{Val})$.

Non-terminaux optionnels Comme suggéré dans la section 3.2, nous pouvons interpréter tout non-terminal optionnel de la forme (A) au moyen de deux productions alternatives. Ainsi, pour tout symbole $A \in \mathcal{N}$, nous ajoutons le symbole (A) à $\mathcal{N}$, et les productions $(A) \to A; \emptyset; \emptyset$ et $(A) \to \bot; \emptyset; \emptyset$ à l'ensemble des productions de G, permettant de réécrire un (A) optionnel en A ou $\bot$, sans contrainte logique ou étiquette de variable associée ($\Lambda = \emptyset$ désignant la fonction d'étiquetage de domaine vide).

Règles lexicales Afin de factoriser un grand nombre de règles lexicales, nous avons proposé d'utiliser des propriétés du lexique comme symboles terminaux, pouvant être implicitement remplacés par n'importe quelle entrée lexicale possédant cette propriété. Nous interprétons donc maintenant toute propriété lexicale $p \in P$ comme un symbole non-terminal ; en ajoutant une règle $p \to (l, P_l); \emptyset; \emptyset$ à l'ensemble des productions pour toute entrée lexicale $(l, P_l) \in \mathscr{L}$ telle que $p \in P_l$ – ces nouvelles productions permettent de réécrire p en une entrée lexicale correspondante. Nous ajoutons également les paires (l, P_l) (les entrées lexicales) à l'alphabet gradué $\mathcal{S}$, en leur donnant l'arité zéro.

Déplacement des contraintes Nous modifions maintenant les contraintes associées aux productions de G afin de ne les faire dépendre que du nœud situé à la racine du membre droit. Pour ce faire, nous modifions chaque formule logique en liant par un quantificateur existentiel toutes les variables libres hormis celle étiquetant la racine du membre droit, et ajoutons la contrainte supplémentaire que les positions que dénotent les variables ainsi quantifiées doivent être situées dans la structure abstraite à une position relative (par rapport à la racine) correspondant à leur position dans l'arbre. La figure 3.8 illustre cette transformation sur une production simple (à gauche) : la partie droite de la figure donne le résultat du déplacement de la contrainte ϕ, à savoir une formule où seule la variable x demeure libre.

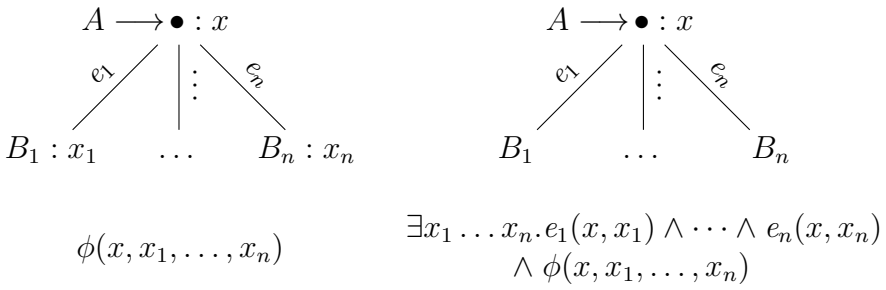


FIGURE 3.8 – Déplacement d'une contrainte vers la racine

Plus précisément, nous modifions la grammaire enrichie G comme suit : dans toute production $A \to t; \Lambda; \Phi$ telle que l'étiquette de la racine est $\Lambda(\varepsilon) =$

x, nous appliquons la séquence de transformations suivante : pour toute position valide $p \in \mathrm{Dom}(t)$ telle que $p \neq \varepsilon$ (excluant la racine), si l'étiquette de p est $\Lambda(p) = x_p$, nous remplaçons chaque formule $\phi \in \Phi$ par $\exists x_p.p(x, x_p) \wedge \phi$. La formule résultante dépend alors d'une variable libre x, qui dénote le nœud correspondant à la racine du membre droit dans la structure abstraite résultante. Nous supposons ici que la racine du membre droit de toute production est étiquetée par une variable ; le cas échéant, ajouter une étiquette x sur la racine si elle en est dénuée ne modifie pas le langage décrit par G.

Construction de la grammaire G' Nous pouvons maintenant construire la grammaire régulière de termes G' esquissée précédemment, qui construit des structures abstraites selon les dérivations permises par la grammaire d'approximation de G, en y incluant explicitement les contraintes qu'elles doivent satisfaire. Cette grammaire est obtenue à partir des productions modifiées de G (par la suite, G désigne implicitement la grammaire obtenue à la suite des transformations ci-dessus), en décorant les symboles terminaux et non-terminaux par des ensembles de contraintes devant être localement satisfaites.

Soit $\mathcal{S}_t = \{\bullet, \bot\} \cup \mathscr{L}$ l'ensemble des symboles terminaux de G et $\mathcal{N} \cup P$ l'ensemble de ses symboles non-terminaux, et soit $\mathcal{F} = \{\Phi \mid (A \to t; \Lambda; \Phi) \in \mathcal{R}\}$ l'ensemble des contraintes apparaissant dans G. Formellement, la grammaire $G' = (\mathcal{N}', \mathcal{S}', \mathcal{P}, S')$, qui produit l'ensemble des termes dérivables décorés par leurs contraintes dans G, est construite comme suit :

- $\mathcal{N}' = \{(A, F) \mid A \in \mathcal{N} \cup P \wedge F \subseteq \mathcal{F}\}$ est l'ensemble des paires formées d'un symbole non-terminal de G et d'un ensemble de contraintes de G.
- $\mathcal{S}' = \{(s, F) \mid s \in \mathcal{S}_t \wedge F \subseteq \mathcal{F}\}$ est l'ensemble des paires formées d'un symbole terminal de G et d'un ensemble de contraintes à satisfaire.
- $S' = (S, \emptyset)$ est le symbole de départ de G, sans contraintes associées.
- $\mathcal{P}$ est l'ensemble des productions de la forme :

$$(A, F) \to (s, F \cup \{\Phi\})\,(t'_1 \ldots t'_n)$$

telles que :

1. $A \to s(t_1 \ldots t_n); \Lambda; \Phi$ est une production enrichie de $\mathcal{R}$.
2. $t'_i = t_i[s \leftarrow (s, \emptyset)]$ pour tout $s \in \mathcal{S}_t \cup \mathcal{N} \cup P$ et tout $i \in [n]$.
3. $F \subseteq \mathcal{F}$.

Les conditions 1, 2 et 3 garantissent que l'ensemble F des contraintes associées au non-terminal A du membre gauche sont reportées sur le symbole s situé à la racine du membre droit, en y ajoutant la contrainte Φ apportée par la production enrichie de $\mathcal{R}$ utilisée ; le symbole s peut être terminal aussi bien que non-terminal. Les symboles des sous-termes éventuels dans le membre droit se voient associer un ensemble de contraintes vide, puisque toutes les contraintes ont été reportées sur la racine au cours de l'étape précédente.

Construction de la formule MSOkS Val Enfin, il nous reste à construire la formule logique Val, qui vérifie simplement que si un symbole à une position x dans un terme est associé à un ensemble de contraintes $F \subseteq \mathcal{F}$, alors la conjonction de toutes les formules de F est vérifiée. La formule résultante est :

$$\text{Val} \overset{def}{\Leftrightarrow} \forall x. \llbracket F \rrbracket (x) \Rightarrow \bigwedge_{\Phi \in F} \Phi(x) \qquad \text{avec :} \qquad \llbracket F \rrbracket (x) = \bigvee_{s \in \mathcal{S}_t} (s, F)(x)$$

Langage résultant Pour finir, le langage régulier de termes associés à G peut maintenant être construit à partir de $\mathcal{L}(G')$ et $\mathcal{L}(\text{Val})$, en considérant l'intersection de ces deux langages (G' assurant le respect des dérivations de G et Val la satisfaction des contraintes logiques associées), puis en effaçant par un morphisme linéaire l'information portant sur les formules logiques des termes du langage résultant (ce qui préserve la régularité). Ce morphisme est un simple réétiquetage, qui peut être défini par $h((s, F)(t_1 \dots t_n)) = s(h(t_1) \dots h(t_n))$.

Le langage $h(\mathcal{L}(G') \cap \mathcal{L}(\text{Val}))$ est alors effectivement l'ensemble des structures des structures abstraites valides sanctionnées par les règles de G.

3.4 Langage concret et linéarisation

Une fois l'ensemble des structures abstraites valides spécifié, il nous reste à décrire la relation entre ces dernières et leurs représentations concrètes, ou *réalisations*. Cette relation, nommée *linéarisation*, associe à toute structure abstraite valide un ensemble de réalisations. Nous la décrirons par l'intermédiaire de règles de construction ajoutées aux productions de la grammaire enrichie : la réalisation associée au non-terminal à gauche d'une production sera presque toujours (hormis dans le cas des requêtes logiques décrites plus bas) obtenue en combinant de diverses façons les réalisations associées aux non-terminaux du membre droit, à la manière des grammaires S-attribuées[Knuth, 1968]. Nous ferons à nouveau usage du langage logique décrit précédemment, pour conditionner des choix entre plusieurs réalisations.

De plus, un même langage abstrait pourra se voir associer plusieurs linéarisations différentes, par exemple dans le but de représenter différents aspects de l'énoncé (forme phonologique, sémantique, *etc.*). Afin d'unifier la présentation des linéarisations et de souligner la similarité avec l'approche des ACG, nous utiliserons le lambda-calcul simplement typé pour construire tous les types de réalisations, qu'il s'agisse de mots, de termes, de formules logiques ou d'opérations sur ces structures.

3.4.1 Règles de linéarisation

Dans leur version la plus simple, nos linéarisations sont décrites par des lambda-termes simplement typés, attachés aux productions de la grammaire

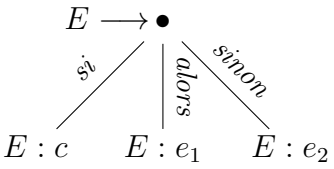
enrichie, et nommés *règles de linéarisation*. Ces lambda-termes décrivent la réalisation associée au non-terminal à gauche de la production. Chacune de leurs variables libres est associée à un non-terminal du membre droit, et dénote la réalisation associée à ce dernier. Puisque toute structure abstraite valide peut être associée à une dérivation selon ces productions, l'ensemble des réalisations d'une structure abstraite se définit naturellement à partir de l'ensemble des dérivations valides de cette structure : une réalisation valide est obtenue en considérant le lambda-terme associé à la production réécrivant le symbole de départ, et en y remplaçant itérativement toute variable associée à un non-terminal par le terme associé à la production utilisée pour réécrire cette occurrence du non-terminal.

Les lambda-termes associés aux productions terminales (construites implicitement à partir du lexique) sont dénotés par le nom l de l'entrée lexicale $(l, P_l) \in \mathscr{L}$, et leur interprétation dépend du langage concret visé. Chaque entrée lexicale pourra ainsi être associée par la suite à une constante ou à un lambda-terme clos représentant, suivant le cas, un mot sur un alphabet, une formule logique dénotant le sens du lexème, ou autre.

Nous établissons ensuite le lien entre les variables libres apparaissant dans les règles de linéarisation et les non-terminaux correspondants en étiquetant les membres droits des productions par des variables, de la même manière que précédemment pour les contraintes logiques. Par économie de notation, nous réutiliserons les mêmes variables que celles apparaissant dans les contraintes logiques ; à ceci près que seules les variables étiquetant des symboles non-terminaux sont susceptibles d'apparaître libres dans les lambda-termes décrivant les réalisations.

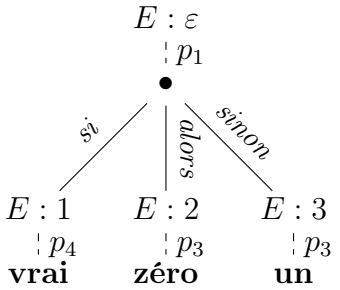
Afin d'illustrer ce fonctionnement la figure 3.10 montre la construction d'un lambda-terme à partir d'un arbre de dérivation, en suivant la règle de linéarisation donnée par la figure 3.9. Le langage concret utilisé est une transcription des expressions du langage abstrait utilisé comme exemple précédemment. Ces expressions sont représentées dans un langage de programmation fonctionnel fictif, utilisant les mots-clés `if`, `then` et `else`, ainsi que les opérateurs et constantes présents dans le lexique $\mathscr{L}$ (donné plus haut par la table 3.1). Le code source d'un programme écrit dans ce langage sera une chaîne de caractères, dénotée par un lambda-terme, suivant la manière évoquée au début de la section 2.2.4. Des réalisations intermédiaires sont attachées aux non-terminaux de la dérivation : dans notre exemple, ceux-ci correspondent exactement aux positions valides dans l'arbre représentant la structure abstraite (ε pour le symbole de départ et 1, 2 et 3 pour les autres). La réalisation de la structure est construite de manière ascendante : nous associons aux non-terminaux situés aux positions 1, 2 et 3 des lambda-termes correspondant aux entrées lexicales qui les réécrivent (respectivement `true`, `zero` et `one`). Puis, nous substituons ces derniers aux variables c, e_1 et e_2 dans la réalisation attachée à la production p_1, pour obtenir la réalisation attachée à la racine – c'est-à-dire au symbole de

départ.



$\lambda x^*.\texttt{if}\ (c\ (\texttt{then}\ (e_1\ (\texttt{else}\ (e_2\ x)))))$

FIGURE 3.9 – Règle de linéarisation



$1 : \texttt{true}\ ;\quad 2 : \texttt{zero}\ ;\quad 3 : \texttt{one}$
$\varepsilon : \lambda x^*.\texttt{if}\ (\texttt{true}\ (\texttt{then}\ (\texttt{zero}\ (\texttt{else}\ (\texttt{one}\ x)))))$

FIGURE 3.10 – Construction de la réalisation d'une structure abstraite

Le lambda-terme obtenu dans l'exemple ci-dessus est immédiatement en forme β-normale, et représente effectivement la chaîne de caractères « `if true then zero else one` ». Le cas échéant, la réalisation associée à une structure correspond au lambda-terme associé à sa racine modulo $=_{\beta\eta}$: pour illustrer ce fait, considérons une autre linéarisation possible pour notre langage, qui évalue directement les expressions du langage abstrait. Nous remplaçons la règle de linéarisation donnée par la figure 3.9 (associée à la production p_1) par : $c\ e_1\ e_2$. Cette règle applique simplement le lambda-terme associé à c aux lambda-termes associés à e_1 et e_2. En outre, nous associons aux entrées lexicales des lambda-termes dénotant des entiers naturels, dits entiers de Church[Church, 1940], et des opérations booléennes sur ces derniers. Les lambda-termes associés aux entrés lexicales devenant respectivement : $\lambda m^{(*\to*)\to*\to*}.\lambda n^{(*\to*)\to*\to*}.m$ (**vrai**), $\lambda f^{*\to*}.\lambda x^*.x$ (**zéro**) et $\lambda f^{*\to*}.\lambda x^*.f\ x$ (**un**). La réalisation associée à la racine ε est alors (en omettant, pour abréger, les annotations de type) :

$$\begin{aligned}(\lambda m.\lambda n.m)(\lambda f.\lambda x.x)(\lambda f.\lambda x.f\ x) &\to_{\beta\eta} (\lambda n.\lambda f.\lambda x.x)(\lambda f.\lambda x.f\ x)\\ &\to_{\beta\eta} \lambda f.\lambda x.x\\ &= \textbf{zéro}\end{aligned}$$

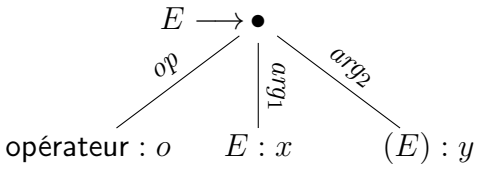
3.4.2 Réalisations multiples et conditions

La relation que nous voulons établir entre structures abstraites et concrètes n'est pas bijective en général. Nous souhaiterions pouvoir construire plusieurs paraphrases issues d'une même structure, et il se peut qu'à l'inverse, certaines structures abstraites n'aient pas de représentation valide dans un certain domaine – par exemple, dans le cas d'une grammaire synchrone multilingue. Afin

de tenir compte de ces faits, nous proposons d'associer un ensemble de lambda-termes représentant les diverses linéarisations possibles à chaque production (*réalisations multiples*), et d'ajouter la possibilité d'attacher à chacun d'entre eux une condition logique, qui doit être satisfaite pour permettre le choix de linéarisation correspondant (*condition de réalisation*).

Nous noterons les réalisations multiples en spécifiant directement l'ensemble des lambda-termes possibles ($\{M_1 \dots M_n\}$). Par facilité de lecture, ces lambda-termes seront énumérés ligne par ligne dans les figures. Les conditions de réalisation seront pour leur part notées en spécifiant une formule logique ϕ dénotant la condition à satisfaire et la réalisation M ainsi conditionnée, séparées par un symbole spécifique : $\longrightarrow$. La réalisation conditionnelle résultante $\phi \longrightarrow M$ se lit « phi permet M » et s'interprète comme « le choix de la réalisation M est permis lorsque la contrainte logique ϕ est vérifiée ».

La figure 3.11 exemplifie ces notations, en étendant la linéarisation proposée par la figure 3.9 à la production enrichie p_2 : la règle correspondante tient compte de l'optionnalité du nœud étiqueté par y dans la production, et produit une réalisation cohérente dans tous les cas. Les constantes `op` et `cp` dénotent respectivement une parenthèse ouvrante et fermante dans le code source.



$$\left\{ \begin{array}{lcl} \mathsf{unaire}(o) & \longrightarrow & \lambda a^*.o\ (x\ a) \\ \neg\mathsf{unaire}(o) & \longrightarrow & \lambda a^*.\mathtt{op}\ (x\ (o\ (y\ (\mathtt{cp}\ a)))) \end{array} \right\}$$

FIGURE 3.11 – Règle de linéarisation avec réalisations multiples et conditions

Remarquons que les conditions logiques de réalisation peuvent également s'appuyer sur les variables attachées aux membres droits des productions enrichies. Les variables libres apparaissant dans ces formules logiques sont interprétées de la même manière que précédemment dans les contraintes de bonne formation. Une réalisation est dite *possible* lorsque sa pré-conditions est localement satisfaite : tout comme pour la validité d'une structure abstraite, cette notion dépend des choix effectués dans la dérivation de la structure abstraite. L'ensemble des réalisations possibles pour une structure abstraite est naturellement l'union des réalisations possibles pour chacune de ses dérivations valides : cet ensemble n'est pas toujours un singleton, et peut éventuellement être vide.

Alternativement, une production $A \to t$ contrainte par un ensemble de formules Φ et munie d'une règle de linéarisation $\{\phi_1 \longrightarrow M_1 \dots \phi_n \longrightarrow M_n\}$ peut être interprétée simplement comme un ensemble de n productions $A \to t$, chacune contrainte par $\Phi \cup \{\phi_i\}$ et munie de la règle de linéarisation M_i.

3.4.3 Requêtes logiques

Finalement, nous souhaitons explorer la possibilité de prendre certaines libertés par rapport aux dérivations lors de la linéarisation, en construisant la réalisation d'une production à partir de réalisations attachées à des positions arbitrairement distantes dans la structure abstraite. Ces positions sont spécifiées au moyen de la même signature logique que précédemment, et les constructions résultantes sont appelées des *requêtes logiques*.

Ces requêtes s'avèrent particulièrement utiles lorsque le regroupement des syntagmes dans une phrase ne coïncide pas avec leurs relations sémantiques ; leur emploi apparaît cependant questionnable, dans la mesure où elles tendent à briser la corrélation existante entre la dérivation d'une structure abstraite et sa réalisation. Nous justifions leur existence par le fait que les contraintes de bonne formation des structure abstraites ont la possibilité de quantifier (grâce aux relations construites à partir d'expressions régulières) sur des nœuds arbitrairement distants de la production associée. Dans de tels cas, la réalisation attachée à ces nœuds distants peut impacter directement la réalisation de cette production.

En d'autres termes, nous considérons que la structure abstraite est soutenue par les contraintes logiques qui lui donnent forme, et non par la seule grammaire d'approximation. Cette situation est illustrée dans notre modélisation du mouvement (page 91), où l'usage de réalisations distantes est corrélé avec des contraintes logiques requérant l'existence des nœuds correspondants. Bien qu'il soit possible d'obtenir dans ce cas les mêmes réalisations en l'absence de requêtes logiques (par l'intermédiaire du lambda-calcul), les règles de linéarisation résultantes sont significativement plus complexes, nuisant à la maintenabilité de la grammaire.

La figure 3.12 illustre l'emploi d'une requête logique pour simplifier la réalisation associée à certaines structures de notre langage. Nous étendons la seconde linéarisation esquissée dans la section 3.4.1, qui évalue les expressions dénotées par nos structures abstraites, en donnant une règle de linéarisation pour la production p_2. Dans les deux premiers cas, cette règle de linéarisation applique la réalisation de l'opérateur aux réalisations de ses opérandes (en tenant compte de l'optionnalité de y). Dans le dernier cas, la réalisation associée à un nœud v est substituée à l'interprétation usuelle : ce choix n'est possible que si le nœud requis v satisfait la pré-condition $\mathit{chemin\_neutre}(r, v)$. Dans le but de rendre immédiatement visible dans les règles de linéarisation le fait qu'une variable x est issue d'une requête logique, nous utiliserons une police distincte ($\mathbf{x}$) pour la mettre en valeur.

La condition logique *chemin_ neutre* utilisée pour requérir le nœud v dénote l'existence d'une série d'opérations arithmétiques ou logiques neutres entre r et v, comme par exemple $1 * x$, $x - 0$, $x \wedge \mathit{vrai}$, *etc.* ; dans un tel cas, l'interprétation de r est identique à celle de v, quelle que soit la distance qui les

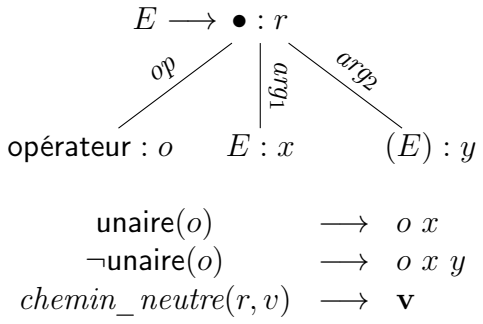


FIGURE 3.12 – Exemple de requête logique

sépare. Une telle condition peut être exprimée au moyen de notre langage logique comme illustré par la table 3.2. Ainsi, dire qu'il existe un chemin neutre entre r et v équivaut à dire que r domine v, et que pour toute expression e (qu'elle soit arithmétique ou logique) située entre r (inclus) et v (exclu), il existe trois nœuds o, a et n désignant respectivement l'opérateur de e et ses deux opérandes ; la notion d'opérande d'une expression étant encodée par la relation $arg(x, y)$, définie immédiatement en dessous. Par suite, a est l'opérande qui domine v, et n doit alors être un élément neutre pour l'opérateur o, ce que dénote la disjonction finale : 1 est neutre pour la multiplication, 0 pour l'addition, vrai pour la conjonction et faux pour la disjonction ; et 0 est neutre à droite d'une soustraction.

$$
\begin{array}{rcl}
\mathit{chemin\_neutre}(r, v) & \triangleq & \mathrm{dom}^+(r, v) \wedge \forall e.\, \mathrm{dom}^*(r, e) \wedge \mathrm{dom}^+(e, v) \Rightarrow \\
& & \exists o, a, n.\, \mathit{op}(e, o) \wedge \mathit{arg}(e, a) \wedge \mathrm{dom}^*(a, v) \wedge \mathit{arg}(e, n) \wedge \\
& & \big(\quad \mathbf{fois}(o) \wedge \mathbf{un}(n) \\
& & \vee \quad \mathbf{plus}(o) \wedge \mathbf{z\acute{e}ro}(n) \\
& & \vee \quad \mathbf{et}(o) \wedge \mathbf{vrai}(n) \\
& & \vee \quad \mathbf{ou}(o) \wedge \mathbf{faux}(n) \\
& & \vee \quad \mathbf{moins}(o) \wedge \mathbf{z\acute{e}ro}(n) \wedge \mathit{arg}_2(e, n) \quad \big) \\
\\
\mathit{arg}(x, y) & \triangleq & \mathit{arg}_1 \mid \mathit{arg}_2 \ (x, y)
\end{array}
$$

TABLE 3.2 – relation entre deux positions séparées par des opérations neutres

Il importe de remarquer que, utilisé sans restriction, ce mécanisme de requêtes logiques permet de décrire des réalisations circulaires, en requérant (directement ou indirectement) la réalisation associée à un nœud pour calculer la réalisation de ce même nœud. Plus généralement, le fait d'employer de telles requêtes, de préférence à une composition linéaire des non-terminaux respectant la structure de la dérivation, induit un risque de sanctionner des réalisations qui ignorent une fraction arbitrairement grande de la structure abstraite ou, à l'inverse, qui dupliquent accidentellement les réalisations associées à certains nœuds.

Ces problèmes peuvent être mitigés en imposant une relation d'ordre sur les positions de la structure abstraite, et en autorisant uniquement dans la réalisation d'une production les requêtes logiques portant sur des positions strictement inférieures à la position courante ; ce processus peut être vu comme comme l'ajout d'un ordre d'évaluation sur les nœuds de la structure abstraite pour la linéarisation. L'ordre partiel découlant des relation de dominance entre les positions du terme constitue un candidat naturel pour une telle relation, éliminant de façon plausible les problèmes de circularité.

3.4.4 Calcul d'une linéarisation

Nous expliquons maintenant comment obtenir une construction effective de toute linéarisation associée à une structure abstraite. Nous nous appuyons pour ce faire sur un ensemble de résultats issus de la littérature portant notamment sur les grammaires de remplacement d'hyper-arêtes (*hyperedge replacement grammars* ou HR, *cf.* Bauderon et Courcelle [1987]) et les grammaires catégorielles abstraites du second ordre ($\mathbf{ACG}(2, n)$).

Une réalisation se construit en partant d'un arbre de dérivation (enrichi par les conditions de bonne formation) associé à une structure abstraite valide. Cet arbre de dérivation est également décoré par les conditions de réalisation qu'il satisfait, selon le même processus que celui décrit dans la section 3.3.3 : les formules logiques sont explicitement inscrites sur les nœuds de l'arbre où elles sont instanciées. Par suite, le mécanisme des requêtes logiques est traité en ajoutant, dans l'arbre résultant, une arête reliant le nœud qui effectue la requête à un de ses descendants susceptible d'être le nœud requis. Le graphe résultant est un graphe orienté sans cycle (ou DAG), représentant un arbre dans lequel certains sous-arbres sont partagés ; le dépliage de ce DAG permet d'obtenir la structure du lambda-terme qui réalise la structure abstraite de la manière attendue. Il suffit alors d'y remplacer chaque nœud interne par un lambda-terme associé par la linéarisation pour obtenir la réalisation de la structure abstraite.

La figure 3.13 illustre le dépliage d'un DAG obtenu en ajoutant une arête correspondant à une requête logique dans une structure abstraite. En haut de la figure se trouve une structure abstraite, décorée par une arête supplémentaire étiquetée « REQ » qui relie deux positions séparées par un chemin neutre. Ce lien correspond à la requête logique $\mathit{chemin\_neutre}(r, v)$ figurant dans la règle de linéarisation donnée par la figure 3.12 et définie dans la table 3.2, et relie le second argument de la racine à une feuille (la requête garantit alors que sémantiquement, ces deux nœuds ont la même valeur). En bas de la figure se trouve le résultat du dépliage de ce DAG : le sous-arbre pointé par l'arête REQ est dupliqué et ajouté, et sa copie a pour nœud parent le nœud à l'origine de la requête. Remarquons que ce mécanisme de dépliage peut en général copier des sous-arbres de taille arbitraire (et que ces derniers peuvent eux-même contenir

des requêtes logiques, traitées de la même manière). La réalisation attendue de cette structure abstraite ignorerait ensuite l'ensemble du sous-arbre dominé par la première arête arg_2, à l'exception de la partie dominée par l'arête REQ : la valeur attribuée au second argument de cette expression est en effet **un**, puisque les opérations intermédiaires dans l'arbre ne modifient pas sa valeur (ce que garantit la formule *chemin_neutre*).

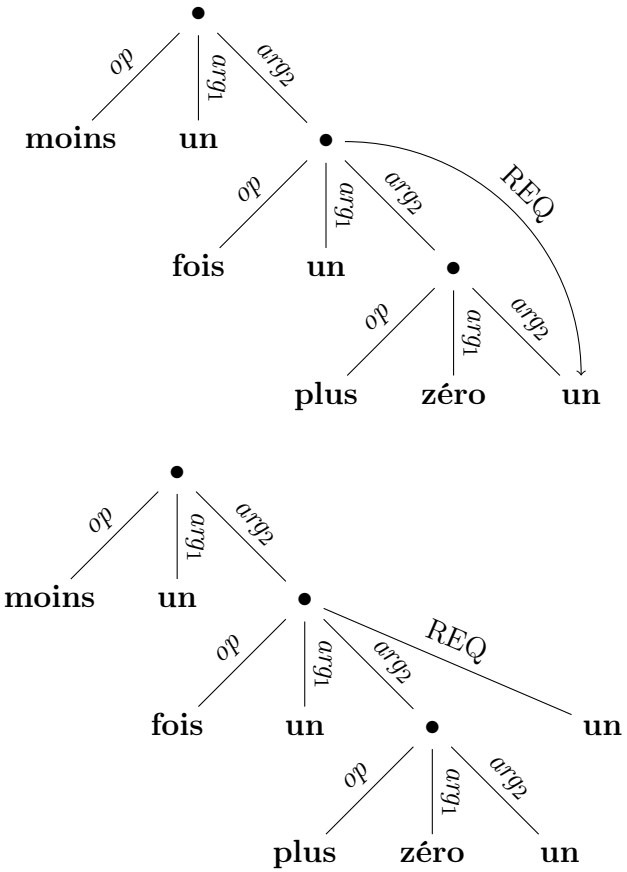


FIGURE 3.13 – Arbre avec partage (DAG) et son dépliage associé

Du point de vue des classes de langages impliquées, l'arbre de dérivation initial et définissable par MSOkS, et appartient donc à un langage régulier d'arbres. Par ailleurs, la classe de langages de graphes produite par les grammaires de remplacement d'hyper-arêtes inclut les langages réguliers d'arbres, et est notamment close par la classe des transductions dites MSO-définissables définies dans [*cf.* Courcelle et Engelfriet, 2011]. L'ajout des arêtes représentant les requêtes logiques constitue une telle transduction ; cependant le dépliage du DAG obtenu est susceptible de produire des copies dépassant le pouvoir d'expression des transductions MSO. La classe de langages d'abres ainsi produite est cependant obtenable par des transducteurs attribués avec anticipation (*look-ahead*)[*cf.* Bloem et Engelfriet, 1998], et donc par des ACG quasi-

linéaires du second ordre [*cf.* Kanazawa, 2009b]. Enfin, l'opération consistant à remplacer les nœuds internes (décorés par les conditions de réalisation qu'ils satisfont) par les lambda-termes que leur associe la linéarisation est un homomorphisme linéaire, et peut aisément être effectuée par une ACG du second ordre.

Il est à remarquer que les ACG quasi-linéaires ne partagent pas les propriétés des ACG décrites dans la section 2.4.2. Cependant, ce surcroît de complexité est essentiellement dû au pouvoir de copie potentiel de l'opération de dépliage de DAG qui permet de traiter les requêtes logiques. Comme les emplois linguistiques réels de ces dernières évitent la duplication des sous-termes impliqués, le coût algorithmique associé peut vraissemblablement être borné.

Sans effectuer en détail la construction que nous avons évoquée, nous détaillons maintenant ses principales étapes et les arguments qui les rendent effectives :

1. L'arbre de dérivation initial, décoré à la fois par les conditions de bonne formation et les conditions de linéarisations instanciées par chaque production, peut être construit en suivant exactement la procédure de compilation décrite dans la section 3.3.3. Celle-ci n'inclut que les conditions de bonne formation, mais les conditions de réalisation s'appuient sur les même variables qui étiquettent les productions, et peuvent donc également être rattachées à la racine des productions et intégrées dans un alphabet gradué pour figurer explicitement dans les termes d'un langage régulier.

2. L'arbre résultant est ensuite complété par des arêtes représentant les requêtes logiques. Celles-ci relient un nœud décoré par une règle de linéarisation comportant une requête logique à un de ses descendants qui satisfait la requête en question. L'ajout de ces arêtes peut être effectué par une transduction MSO-définissable, qui vérifie les informations présentes sur le nœud source, la satisfaction de la requête par le nœud cible (l'ensemble des requêtes logiques dans la linéarisation d'une grammaire est fini), et la relation de domination entre ces deux nœuds.

   De plus, l'emploi de paramètres dans le schéma de définition de cette transduction [*cf.* Courcelle et Engelfriet, 2011, p. 505] permet de ne sélectionner par la suite que les DAG où les requêtes logiques sont fonctionnelles (celles où le nœud qui effectue la requête est associé à un unique nœud requis parmi toutes les cibles possibles). L'ensemble de graphes résultants de cette transduction est définissable par une grammaire de remplacement d'hyper-arêtes.

3. Le dépliage des DAG résultants produit un langage d'arbres, définissable par une grammaire attribuée comme expliqué plus haut. Ces arbres reflètent la structure des lambda-termes qui sont des réalisations valides pour la structure d'origine.

4. Un réétiquetage de ces arbres permet de remplacer un nœud étiqueté par une condition de réalisation ϕ_i par son lambda-terme associé M_i. Le lambda-terme obtenu en considérant le terme M_i de chaque nœud, en y abstrayant les variables libres correspondant à ses arguments ou à ses requêtes logiques, et en l'appliquant aux lambda-termes associés à ses descendants immédiats forme alors une réalisation valide de la structure abstraite d'origine.

L'ensemble des réalisations ainsi décrites est donc obtenu par réétiquetage d'un langage d'arbres (de degré borné) défini par un transducteur attribué avec anticipation. Il est par conséquent définissable par une grammaire catégorielle abstraite quasi-linéaire du second ordre employant une signature de termes. Cette classe de langages est à son tour décidable ; et en l'absence de copie dans l'emploi des requêts logique, la linéarité de l'ACG résultante rend son langage reconnaissable en temps polynomial par rapport à la taille de l'énoncé, comme montré par Salvati [2005]. Il convient toutefois d'observer que, même en l'absence de copies arbitraires, la taille de la grammaire obtenue par ce biais sans optimisation, rend vraisemblablement l'analyse impraticable pour des applications de TAL. Nous reviendrons sur ce point en fin de chapitre.

3.5 Langage de macros pour les linéarisations

Plusieurs des linéarisations les plus complexes apparaissant dans nos modélisations contiennent des réalisations alternatives qui varient subtilement les unes par rapport aux autres (c'est en particulier le cas pour le traitement sémantique de l'antécédence, détaillé page 95). Dans ces cas, les lambda-termes dénotant les réalisations alternatives ne diffèrent que par un sous-terme (ou un contexte) de taille réduite, le reste demeurant identique entre tous les cas. Par souci de concision, nous souhaitons rendre explicites les généralisations que ces répétitions suggèrent, en factorisant les termes ou contextes identiques.

À cet effet, nous proposons un langage de macros mêlant de façon transparente les lambda-termes issus du langage concret avec les réalisations multiples et conditions logiques introduits dans la section 3.4.2, et proposons un système de réécriture permettant d'interpréter les termes résultants comme des ensembles de termes conditionnés, similaires à ceux de la section précédente.

3.5.1 Définition et usage

La réalisation d'une production sera maintenant décrite par des lambda-termes simplement typés sur la signature représentant le langage concret, pouvant inclure des conditions logiques et des ensembles de réalisations alternatives. Nous spécifions maintenant la syntaxe de ces nouvelles règles de linéarisation :

Définition 3.5. Soit $\mathcal{C}$ l'ensemble des constantes (typées) formant la signature du langage concret visé et $\mathcal{T}$ l'ensemble de ses types simples, puis $\mathcal{X} \times \mathcal{T}$ un ensemble dénombrable de variables typées, et $\mathcal{F}$ l'ensemble des conjonctions de formules logiques apparaissant dans les règles de linéarisation de la grammaire.

Le nouvel ensemble $\mathscr{R}$ des *règles de linéarisation* est formé à partir de celui des lambda-termes sur la signature $\mathcal{C}$, auxquels nous ajoutons deux constructions supplémentaires pour les réalisations conditionnelles et alternatives :

— Si $c \in \mathcal{C}$, alors $c \in \mathscr{R}$ (*constante*).
— Si $x^\alpha \in \mathcal{X} \times \mathcal{T}$, alors $x \in \mathscr{R}$ (*variable*).
— Si $M \in \mathscr{R}$, $x^\alpha \in \mathcal{X} \times \mathcal{T}$, alors $(\lambda x^\alpha.M) \in \mathscr{R}$ (*abstraction*).
— Si $M, N \in \mathscr{R}$, alors $(MN) \in \mathscr{R}$ (*application*).
— Si $M \in \mathscr{R}$ et $\phi \in \mathcal{F}$, alors $(\phi \longrightarrow M) \in \mathscr{R}$ (*condition*).
— Si $M_1, \ldots, M_n \in \mathscr{R}$, alors $\{M_1 \ldots M_n\} \in \mathscr{R}$ (*ensemble*).

Une règle de linéarisation ne contenant que des constantes, variables, abstractions et applications est qualifiée de *pure*.

Les deux dernières constructions de la définition précédente dénotent respectivement les réalisations conditionnelles et les réalisations alternatives. Leur interprétation découle naturellement de celle donnée plus haut pour les règles de linéarisation : l'interprétation d'une condition $\phi \longrightarrow M$ est que le sous-terme M est une réalisation possible si et seulement si ϕ est satisfaite ; et celle d'un ensemble $\{M_1 \ldots M_n\}$ est que chaque M_i est une réalisation possible pour $i \in [n]$. Remarquons que l'ensemble $\mathscr{R}$ résultant inclut en particulier les règles de linéarisation simples présentées dans la section 3.4.2, qui se présentent comme des ensembles de conditions portant sur des règles pures.

Afin d'illustrer l'usage de cette syntaxe, nous considérons tout d'abord le cas où l'on cherche à factoriser un contexte identique entre plusieurs alternatives, et où seul un sous-terme varie d'un cas à l'autre. Si C est le contexte à factoriser et $S = \{M_1 \ldots M_n\}$ l'ensemble de ses sous-termes possibles, la règle de linéarisation correspondante est simplement $C[S]$. Par exemple, la règle de linéarisation suivante permet d'employer au choix n'importe lequel des trois symboles $\times$, $*$ ou . pour construire la réalisation du produit de deux termes x et y :

$$\mathbf{fois}(o) \longrightarrow x \left\{ \begin{array}{c} \times \\ * \\ . \end{array} \right\} y$$

Elle abrège, dans la notation de la section précédente, la règle de linéarisation suivante :

$$\left\{ \begin{array}{lcl} \mathbf{fois}(o) & \longrightarrow & x \times y \\ \mathbf{fois}(o) & \longrightarrow & x * y \\ \mathbf{fois}(o) & \longrightarrow & x \ . \ y \end{array} \right\}$$

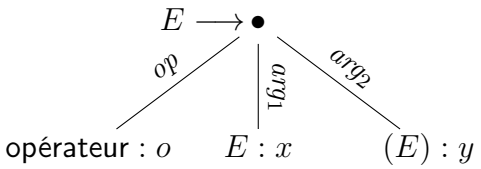
Par suite, dans le cas où l'élément à factoriser est un sous-terme, nous exploitons la sémantique usuelle du lambda-calcul pour en revenir au cas précédent. Ainsi, la règle $S\ N$ où $S = \{\phi_1 \longrightarrow \lambda x.M_1 \ldots \phi_n \longrightarrow \lambda x.M_n\}$ permet de

factoriser N dans chacun des termes M_i. Cette règle s'interprète, en concluant par une β-réduction, en $\{\phi_1 \longrightarrow M_1[x \leftarrow N] \ldots \phi_n \longrightarrow M_n[x \leftarrow N]\}$ (en supposant que x est une variable fraîche lors de la factorisation). Afin de ne pas alourdir nos réalisations avec de multiples abstractions toutefois, nous abrégerons par la suite la construction précédente en utilisant la notation :

$$\left\{ \begin{array}{c} \phi_1 \longrightarrow M_1 \\ \ldots \\ \phi_n \longrightarrow M_n \end{array} \right\} \textit{où } x = N$$

De plus, dans ces cas, x sera usuellement un nom de variable décrivant la fonction du terme N dans la réalisation. Cette possibilité de nommer les sous-termes factorisés constitue un avantage supplémentaire pour la modélisation linguistique et le développement de grammaires, en offrant l'opportunité d'expliciter la généralisation ainsi décrite.

La figure 3.14 illustre un emploi de ce mécanisme en proposant une règle pour le parenthésage des expressions, améliorant la règle proposée plus haut par la figure 3.11. Elle autorise l'omission des parenthèses autour d'un opérateur dont la priorité est forte (priorité $+$), tout en permettant dans tous les cas une réalisation incluant les parenthèses (dénotées par les constantes op et cp). L'expression elle même, incluant l'opérateur et ses opérandes, est factorisée par la variable *expr* entre ces deux cas, évitant la répétition de l'ensemble de réalisations qui apparaît à droite du signe $=$.



$$\left\{ \begin{array}{lcl} \textit{Vrai} & \longrightarrow & \lambda a^*.\mathsf{op}\ (\textit{expr}\ (\mathsf{cp}\ a)) \\ \mathsf{priorité}\ +(o) & \longrightarrow & \textit{expr} \end{array} \right\}$$

$$\textit{où } \textit{expr} = \left\{ \begin{array}{lcl} \mathsf{unaire}(o) & \longrightarrow & \lambda a^*.o\ (x\ a) \\ \neg\mathsf{unaire}(o) & \longrightarrow & \lambda a^*.x\ (o\ (y\ a)) \end{array} \right\}$$

FIGURE 3.14 – Ajout d'une règle de parenthésage des expressions

Enfin, nous emploierons parfois l'expression « *sinon* » en guise de condition logique. Cette condition particulière n'apparaîtra qu'en tant que dernier élément dans un ensemble de conditions : elle est associée à une linéarisation « par défaut », sélectionnée lorsqu'aucune autre condition de cet ensemble n'est vérifiée. En d'autres termes, *sinon* s'interprète comme la conjonction des négations

des autres conditions logiques de l'ensemble, comme illustré ci-dessous :

$$\left\{\begin{array}{lcl} \phi_1 & \longrightarrow & M_1 \\ & \dots & \\ \phi_n & \longrightarrow & M_n \\ sinon & \longrightarrow & N \end{array}\right\} \quad \Leftrightarrow \quad \left\{\begin{array}{lcl} \phi_1 & \longrightarrow & M_1 \\ & \dots & \\ \phi_n & \longrightarrow & M_n \\ \neg\phi_1 \wedge \dots \wedge \neg\phi_n & \longrightarrow & N \end{array}\right\}$$

3.5.2 Interprétation

Nous interpréterons les nouvelles règles de linéarisation de $\mathscr{R}$ en deux étapes : tout d'abord, nous réécrirons ces règles en interprétant les conditions et les ensembles ($\longrightarrow$ et $\{\}$) qu'elles contiennent, jusqu'à obtenir des termes en forme normale. Ces derniers auront la forme $\{\phi_1 \longrightarrow M_1 \dots \phi_n \longrightarrow M_n\}$ où chaque M_i est un terme pur (ne contenant ni ensemble, ni condition). La règle de linéarisation résultante sera ensuite interprétée de la même manière que dans la section précédente.

Nous détaillons maintenant le système de réécriture permettant d'interpréter les constructeurs $\longrightarrow$ et $\{\}$ dans les règles de linéarisation. Celui-ci se compose de deux relations $\rightarrow_C$ (pour les conditions logiques) et $\rightarrow_E$ (pour les ensembles), décrites respectivement par les figures 3.15 et 3.16. Intuitivement, ces systèmes permettent de faire « remonter » les constructeurs qu'ils réécrivent vers la racine du lambda-terme qui les inclut, en respectant l'interprétation suggérée dans la section 3.5.1.

$$\frac{\lambda x.\phi \longrightarrow M}{\phi \longrightarrow \lambda x.M}\, abs \qquad \frac{\phi \longrightarrow \psi \longrightarrow M}{\phi \wedge \psi \longrightarrow M}\, cond$$

$$\frac{(\phi \longrightarrow M)N}{\phi \longrightarrow MN}\, app,g \qquad \frac{M(\phi \longrightarrow N)}{\phi \longrightarrow MN}\, app,d$$

FIGURE 3.15 – Système de réécriture $\rightarrow_C$

Ainsi, dans le cas de $\rightarrow_C$, l'application ou l'abstraction d'un terme réalisé sous une certaine condition est elle-même réalisée sous cette condition (règles *abs*, *app*, *g* et *app*, *d*), et un terme réalisé sous deux conditions ϕ et ψ est réalisé sous la conjonction $\phi \wedge \psi$ de ces deux conditions (règle *cond*).

De même, la relation $\rightarrow_E$ dénote que l'application ou l'abstraction d'un ensemble de réalisation s'interprète comme l'ensemble des applications ou abstractions de chaque réalisation possible (règles *abs*, *app*, *g* et *app*, *d*). De façon transparente, un ensemble de réalisations conditionné par ϕ équivaut à l'ensemble de ces réalisations conditionnées chacune par ϕ (règle *cond*). Pour des raisons techniques qui seront apparentes dans la prochaine section, nous souhaitons également également exprimer le fait qu'un singleton équivaut au terme qu'il contient (règle *sgl*). Enfin, un ensemble de réalisations dont l'une

$$\frac{\lambda x.\,\{M_1 \dots M_n\}}{\{\lambda x.M_1 \dots \lambda x.M_n\}}\ abs \qquad \frac{\phi \longrightarrow \{M_1 \dots M_n\}}{\{\phi \longrightarrow M_1 \dots \phi \longrightarrow M_n\}}\ cond$$

$$\frac{\{M_1 \dots M_m\}\, N}{\{M_1 N \dots M_m N\}}\ app, g \qquad \frac{M\,\{N_1 \dots N_n\}}{\{M N_1 \dots M N_n\}}\ app, d$$

$$\frac{\{M\}}{M}\ sgl \qquad \frac{\{M_1 \dots M_{i-1}, \{M_{i,1} \dots M_{i,m}\}, M_{i+1} \dots M_n\}}{\{M_1 \dots M_{i-1}, M_{i,1} \dots M_{i,m}, M_{i+1} \dots M_n\}}\ ens$$

FIGURE 3.16 – Système de réécriture $\rightarrow_E$

des alternatives est un ensemble de réalisations s'interprète directement comme l'union de toutes les réalisations possibles (règle *ens*).

Enfin, la dernière composante de cette interprétation est la β-réduction usuelle sur les lambda-termes. Nous faisons ici le choix d'adopter une sémantique d'appels par valeur lors de la β-réduction : une application dont le membre gauche est une abstraction ne constitue un β-redex que lorsque son argument (à droite) est une règle de linéarisation pure, ne contenant ni ensemble ni condition.

3.5.3 Propriétés

Nous notons $\rightarrow_{\beta CE}$ le système de réécriture obtenu par combinaison de $\rightarrow_C$ et $\rightarrow_E$ avec la β-réduction usuelle restreinte décrite ci-dessus. Nous démontrons plusieurs propriétés de ce système de réécriture permettant d'obtenir l'interprétation souhaitée. Tout d'abord, nous montrons qu'il est localement confluent – c'est-à-dire que tout choix entre deux réécritures possibles dans un terme produit une paire de termes qui est dite *joignable* ; une paire de termes (R_1, R_2) étant joignable pour une relation $\rightarrow$ si il existe deux séquences de réécritures permettant de revenir au même terme : $R_1 \xrightarrow{*} R'$ et $R_2 \xrightarrow{*} R'$. Nous montrons ensuite que $\rightarrow_{\beta CE}$ est fortement normalisant, c'est-à-dire qu'il n'existe pas de terme pouvant être réécrit indéfiniment. Comme pour le lambda-calcul typé (*cf.* 2.2.3), la conjonction de ces deux propriétés induit la convergence de $\rightarrow_{\beta CE}$, c'est-à-dire l'existence et l'unicité d'une forme normale pour chaque règle de linéarisation. En outre, le système $\rightarrow_{\beta CE}$ possède également sous une forme faible la propriété de réduction du sujet : le type d'une réalisation (ou d'un fragment de réalisation) n'est pas modifié lors de sa réécriture.

Système de typage Afin de montrer ces résultats, nous dotons les termes de $\mathscr{R}$ d'un type selon le système donné par la figure 3.17. Ce système préserve les types classiques des lambda-termes à travers les conditions, et permet de typer un ensemble de termes de type uniforme ; en outre, un ensemble de termes de

types hétérogènes peut également être typé, au moyen d'un type atomique dédié ω, n'appartenant pas à $\mathcal{T}$. Cette dernière règle de typage est absorbante, en ce sens qu'un terme contenant un sous-terme de type ω ne peut qu'être un ensemble typé par ω (reflété par la condition $\alpha, \beta \neq \omega$, à l'exclusion des α_i).

$$\frac{x^\alpha \in \mathcal{X} \times \mathcal{T}}{x : \alpha}\, var \qquad \frac{M : \beta}{\lambda x^\alpha . M : \alpha \to \beta}\, abs \qquad \frac{M : \alpha \to \beta \quad N : \alpha}{MN : \beta}\, app$$

$$\frac{x \in C}{c : \tau(c)}\, cst \qquad \frac{M : \alpha \quad \phi \in \mathcal{F}}{\phi \longrightarrow M : \alpha}\, cond \qquad \frac{M_1 : \alpha \quad \cdots \quad M_n : \alpha}{\{M_1 \dots M_n\} : \alpha}\, ens$$

$$\frac{M_1 : \alpha_1 \quad \cdots \quad M_n : \alpha_n}{\{M_1 \dots M_n\} : \omega}\, ens, \omega$$

FIGURE 3.17 – Système de typage pour les règles de linéarisation ($\alpha, \beta \neq \omega$)

Le type ω reflète la possibilité de fournir des règles de linéarisation polymorphes : une production peut se voir associer une réalisation de type τ si une condition ϕ est vérifiée, et une autre réalisation de type σ si ψ est vérifiée ; c'est par exemple le cas lors de la linéarisation en sémantique des proposition relatives ou contrôlées dans le chapitre 4. Implicitement, une telle règle de linéarisation n'est acceptable que si $\phi \wedge \psi \Rightarrow \mathit{Faux}$: le non-terminal à gauche a alors le type τ dans le contexte de ϕ, et σ dans le contexte de ψ. Par suite, toute production dont le membre droit inclut ce symbole et dont les conditions sont compatibles avec ϕ et ψ doit tenir compte des différents types possibles, afin de ne produire que des termes bien typés. Le bon typage de toutes les réalisations possibles pour une structure abstraite donnée n'est pas garanti par le système décrit ici : par simplicité, nous avons fait le choix de rejeter comme invalides les réalisations qui ne sont pas typables.

Une autre possibilité aurait été de restreindre l'ensemble des grammaires possibles à celles dont les contraintes et règles de linéarisation ne sanctionnent que des réalisations bien typées. Compte tenu des avantages apportés par l'emploi de règles de linéarisation de type polymorphe (ω), ce choix requerrait l'emploi d'annotations de typage globales sur les non-terminaux, conditionnées par des formules logiques. Le système résultant n'entre pas dans le cadre de cette thèse, mais constituerait néanmoins un objet d'étude intéressant.

Réduction du sujet Nous commençons par montrer une forme restreinte de réduction du sujet, garantissant que la réduction d'un terme par $\to_{\beta CE}$ préserve son type. Nous qualifions cette propriété de restreinte, car elle repose sur l'emploi de variables typées à la Church : ainsi, une règle de linéarisation

peut contenir deux variables libres x^τ et x^σ, faisant toutes deux référence à la réalisation associée à un unique symbole non-terminal du membre droit. Une telle règle peut ne pas poser de problème, si x^τ et x^σ apparaissent sous deux conditions logiques distinctes (par exemple $\{\phi_1 \longrightarrow x^\tau; \phi_2 \longrightarrow x^\sigma\}$), mais est inacceptable si une des alternatives de réalisation combine les deux usages (par exemple $\phi \longrightarrow f\ x^\tau\ x^\sigma$).

Les termes contenant des typages incohérents seront filtrés ultérieurement, après la mise en forme normale des règles de linéarisation. Moralement, la propriété de réduction du sujet montrée ici garantit que le type d'un terme impliqué dans une règle de linéarisation n'est pas *modifié* lors de l'interprétation ; celle-ci peut cependant révéler des sous-termes incohérents, lesquels seront filtrés par la suite.

Théorème 3.1. *Si une règle de linéarisation $R \in \mathscr{R}$ est typable par $R : \alpha$ avec $\alpha \in \mathcal{T}$ et que $R \xrightarrow{*}_{\beta CE} R'$, alors $R' : \alpha$.*

Démonstration. Par induction sur la longueur de la réduction, nous montrons simplement ce résultat pour $R \rightarrow_{\beta CE} R'$.

Dans le cas où $R \rightarrow_\beta R'$, la démonstration suit le même schéma que celle de la réduction du sujet pour le lambda-calcul simplement typé.

Pour chacune des des règles $\rightarrow_C$ ou $\rightarrow_E$, le résultat est immédiat à partir de la forme des règles (voir figures 3.15, 3.16 et 3.17). Observons que la condition $\alpha \in \mathcal{T}$ est nécessaire : la règle *sgl* de $\rightarrow_E$ ne permet pas de conserver le typage $\{M\} : \omega$ si M n'est pas de type ω (elle préserve en revanche la possibilité de lui donner le type de M). □

Normalisation forte Nous montrons par la suite que toute séquence de réécritures par $\rightarrow_{\beta CE}$ est de longueur finie, par un argument similaire à celui de la normalisation forte du lambda-calcul typé. La preuve, assez longue, de ce résultat se trouve dans la deuxième section de l'annexe A.

Théorème 3.2. *Le système de réécriture $\rightarrow_{\beta CE}$ est fortement normalisant.*

Démonstration. Voir annexe A.2. □

Confluence locale Nous démontrons maintenant la confluence (locale) du système de réécriture interprétant les règles de linéarisation combiné à la β-réduction classique.

Lemme 3.3. *Le système de réécriture $\rightarrow_{\beta CE}$ est localement confluent.*

Démonstration. La confluence locale de $\rightarrow_\beta$ pour nos termes s'obtient de la même manière que pour le lambda-calcul (théorème 2.1).

Par suite, la confluence locale pour le système entier s'obtient en identifiant toutes les paires dites *critiques* de $\rightarrow_{\beta CE}$, formées lorsqu'un fragment d'un

terme de $\mathscr{R}$ peut se réécrire selon deux règles différentes, et en montrant qu'elles sont joignables. La confluence locale s'ensuit immédiatement[*cf.* Kirchner et Kirchner, 2006, p. 200].

Les paires critiques du système $\rightarrow_C$ sont dues à un choix entre *cond* et n'importe quelle règle de $\rightarrow_C$ ou à un choix entre app, g et app, d. Similairement, les paires critiques de $\rightarrow_E$ impliquent la règle *ens* avec n'importe quelle règle, ou les règles app, g et app, d ; la règle *sgl* entre également en conflit (immédiatement résolu) avec les autres règles du système. Ensuite, la combinaison de ces deux systèmes produit des paires critiques entre la règle *cond* de $\rightarrow_E$ et n'importe quelle règle de $\rightarrow_C$, ainsi qu'entre les règles app, g et app, d de ces deux systèmes. Enfin, la combinaison avec $\rightarrow_\beta$ introduit deux nouvelles paires critiques, entre la β-contraction et les règles *abs* de chacun des systèmes.

Observons que sémantique d'appels par valeur donnée plus haut pour la β-réduction interdit à un ensemble ou une condition d'appartenir au membre droit d'un β-redex, ce qui prévient l'existence de paires critiques non-joignables entre β et app, d.

La liste exhaustive des paires critiques de $\rightarrow_{\beta CE}$, ainsi que les stratégies permettant de les joindre, est donnée par l'annexe A.1. □

Forme normale

Théorème 3.4. *Le système de réécriture $\rightarrow_{\beta CE}$ induit une forme normale unique $|R|_{\beta CE}$ pour tout terme $R \in \mathscr{R}$.*

Démonstration. L'existence d'une forme normale est due au théorème 3.2, et son unicité est alors une conséquence immédiate du lemme 3.3. □

Conséquences Les propriétés listées dans cette section entraînent que les règles de linéarisation construites au moyen de notre langage de macros s'interprètent comme des ensembles de conditions de règles de linéarisation pures, tels que décrites précédemment. En effet, la mise en forme normale impose que tout terme contenu dans un ensemble est une condition portant sur un terme pur, ou un terme pur (sans quoi il contiendrait un redex selon le système $\rightarrow_{CE}$). L'unicité de cette forme normale et la normalisation forte impliquent que toute stratégie de réduction conduit à la même interprétation.

En outre, la duplication des termes et formules logiques apportée par les règles de réduction de $\rightarrow_E$ permet à nos règles de linéarisation de représenter de façon compacte des règles pures contenant de nombreux sous-termes et/ou formules logiques identiques.

3.6 Conclusion

À l'issue de ce chapitre, nous disposons d'un formalisme permettant de décrire des langages, formé de deux composantes essentielles.

La première est un mécanisme de description du langage à un niveau abstrait. La structure des énoncés résulte d'une grammaire régulière de termes, dite grammaire support, qui décrit grossièrement un sur-ensemble du langage d'arbre visé. Les productions de la grammaire support sont ensuite étiquetées et décorées par des formules logiques, lesquelles s'appuient sur une signature du premier ordre sur les termes enrichie par des relations régulières sur les chemins, à des fins de description linguistique. Ces formules sont instanciées et interprétées comme des contraintes de grammaticalité, filtrant les structures abstraites ne satisfaisant pas leurs contraintes associées. La signature sur laquelle elles s'appuient peut être exprimée à partir d'une signature du second ordre monadique à k successeurs, ce qui nous a permis de donner une caractérisation effective du langage régulier de termes résultant. Il est à noter que l'automate correspondant est susceptible d'être de très grande taille, même si les contraintes et la grammaire support qui le caractérisent sont assez brèves, en raison de l'expressivité de la logique utilisée pour formuler les contraintes. Cet obstacle peut toutefois être mitigé en substituant au processus de compilation naïve décrit dans la section 3.3.3 les techniques évoquées par Morawietz et Cornell [1997], en s'appuyant sur des outils logiciels dédiés tels que MONA [Klarlund et Møller, 2001].

La seconde composante de notre formalisme consiste en un processus de linéarisation, qui calcule une représentation concrète d'un énoncé du langage abstrait. Cette représentation, nommée réalisation, est obtenue à l'aide du lambda-calcul simplement typé ; elle consiste en un lambda-terme qui dénote un aspect sémantique ou syntaxique de l'énoncé. Ce terme est calculé de manière ascendante, en suivant la dérivation de la structure abstraite qu'offre la grammaire support, par le biais de règles de linéarisations rattachées à ses productions. Celles-ci combinent généralement les réalisations associées aux non-terminaux du membre droit (à la manière d'une grammaire d'attributs), et exploitent le langage logique que nous avons proposé pour les contraintes de grammaticalité, afin de sélectionner certaines réalisations en fonction du contexte de la structure abstraite. Nous avons également proposé un langage de macros étendant le lambda-calcul simplement typé, pour décrire de manière compacte des règles de linéarisation dont certains fragments dépendent de préconditions logiques. Nous avons ensuite montré son bon fonctionnement dans le cadre d'une sémantique d'appels par valeur. Enfin, nous avons introduit un mécanisme de requêtes logiques, permettant le déplacement à travers la structure abstraite de composants de sa réalisation : ces requêtes s'appuient une fois encore sur notre langage logique, et permettent de s'échapper partiellement du cadre offert par les dérivations de la grammaire support.

Les langages ainsi décrits sont effectivement des ACG du second ordre produisant des lambda-termes ; lesquels sont en correspondance immédiate avec les structures qu'ils représentent (mots, formules logiques, *etc.*). Bien que décidables en temps polynomial par rapport à la taille d'un énoncé, le problème

de la taille des représentations des langages ainsi obtenus se pose une fois de plus. Les résultats théoriques qui permettent leur construction n'offrent pas de garantie explicite en termes de complexité algorithmique au delà de la décidabilité. Cependant, l'existence de modèles approximatifs des langages naturels dans d'autres formalismes algorithmiquement abordables, ainsi que le nombre limité de concepts linguistiques à décrire, permettent de supposer qu'il est possible de fournir un modèle acceptable de la compétence linguistique sans requérir les pires cas de complexité atteignables par le formalisme. En outre, notre emploi de résultats et d'outils simples issus de l'informatique fondamentale offre un avantage supplémentaire, à savoir que la mise au point de techniques de compilation dont la complexité se limite à celle qui est réellement requise par une grammaire (et sa linéarisation) est étroitement liée aux connaissances existantes sur la structure et la complexité propres de la logique et du lambda-calcul.

Observons finalement que, bien que nous ayons choisi d'illustrer dans ce chapitre le fonctionnement du formalisme par un langage artificiel basique, certains des choix de conception effectués répondent directement à des besoins linguistiques spécifiques : c'est en particulier le cas des requêtes logiques, introduites pour faciliter la modélisation des phénomènes de mouvement. D'autres pans du formalisme, comme le langage de macros des règles de linéarisation, visent à faciliter l'ingénierie grammaticale. L'objectif directeur de ces choix est la capacité de proposer des descriptions concises et de haut niveau de divers phénomènes linguistiques, dans la continuité de l'emploi de notre langage logique sur les structures abstraites. Afin de confronter cet objectif au formalisme que nous avons décrit jusque là, le prochain chapitre illustre le fonctionnement de nos outils de description à travers un ensemble de modélisations linguistiques, incluant celles qui ont guidé notre travail de conception.

Chapitre 4

Modélisations linguistiques

Nous souhaitons maintenant mettre en œuvre le formalisme décrit dans le chapitre précédent pour modéliser divers phénomènes linguistiques. Cet exercice constitue essentiellement une mise à l'épreuve des outils proposés plus haut face à une tâche de modélisation réelle dans le domaine de la linguistique formelle. Nous nous appuierons sur des descriptions existantes et matérialiserons les concepts sur lesquels elles s'appuient au moyen de formules logiques, que nous utiliserons pour décrire le langage abstrait des énoncés valides. Notre modélisation permettra également de reconstruire la forme phonologique de ces énoncés, ainsi qu'une représentation de leur sens.

Nous produirons ainsi une grammaire munie de contraintes, et deux linéarisations associées, en syntaxe et en sémantique. Cette grammaire couvrira initialement (section 4.2) un modèle simple de la phrase. Nous étendrons ensuite ce modèle en y incluant un traitement de l'accord en genre, nombre, et personne, afin d'illustrer comment l'emploi de propriétés lexicales et de contraintes logiques se substitue à la multiplication de symboles non-terminaux, ou à l'emploi de structures de traits (section 4.3). Nous inclurons ensuite dans notre grammaire des propositions subordonnées, complétives et relatives (section 4.4). L'emploi des pronoms relatifs dans ces dernières entraîne usuellement un traitement des dépendances à longue distance, incluant des contraintes dites d'îlot : nous traiterons ces phénomènes par le biais de contraintes logiques intégrant des expressions régulières. En outre, le phénomène de mouvement associé à l'antéposition du pronom sera traité au moyen de requêtes logiques. Enfin, nous élargirons la grammaire par l'ajout de subordonnées infinitives dépendant de verbes dits à contrôle (section 4.5). Outre leur traitement sur le plan sémantique, nous proposerons pour ces derniers plusieurs linéarisations synchrones en syntaxe, soulignant les différences et les similarités dans la construction de propositions subordonnées complexes entre l'anglais, l'allemand et le néerlandais. Ces linéarisations parallèles illustreront un intérêt de la séparation explicite entre la structure d'un énoncé et sa réalisation, ainsi qu'un exemple d'emploi du lambda-calcul pour produire des réalisations

échappant à une analyse hors-contexte directe (dans le cas de l'ordre des mots en néerlandais).

4.1 Langages abstrait et concrets

Nous présentons maintenant la structure du langage abstrait décrit par notre grammaire, ainsi que celle des deux langages concrets évoqués plus haut (forme phonologique et représentation sémantique). Comme décrit dans les chapitres précédents, ces derniers seront des langages de lambda-termes construits sur une signature appropriée, représentant respectivement des séquences de mots et des formules logiques.

Langage abstrait Comme dans l'exemple du chapitre 3, notre langage abstrait sera un ensemble d'arbres dénotés par des termes, dont les nœuds internes sont construits au moyen d'un symbole unique $\bullet$ muni d'une arité fixée, et dont les feuilles seront des éléments du lexique. Les arêtes seront comme auparavant nommées, chaque étiquette d'arête correspondant à un entier de $[\rho(\bullet)]$. En particulier, la notion de tête syntagmatique sera représentée par une étiquette *tête* ; les autres étiquettes d'arêtes utilisées initialement étant *sujet*, *objet* et *déterm*, dénotant respectivement la relation entre un syntagme et son sujet, son objet et son déterminant.

Le langage que décriront ces arbres coïncide essentiellement avec la notion de syntaxe profonde d'une langue décrite par Chomsky [1957] (*deep structure*). Notons que le regroupement des syntagmes dans une structure abstraite ne correspond pas toujours à celui qui apparaît dans sa forme phonologique, bien qu'ils soient en pratique souvent similaires. Une exception notable à cette corrélation s'observe dans l'antéposition des pronoms relatifs et interrogatifs, susceptibles de traverser un grand nombre de subordonnées enchâssées ; une autre exception apparaissant dans nos modélisations sera l'ordre des mots dans les dépendances croisées en série, en néerlandais.

Les feuilles de nos structures abstraites seront des entrées lexicales, représentées par un mot associé à un ensemble de propriétés par un lexique, tout comme dans l'exemple du chapitre précédent. La table 4.1 présente quelques entrées possibles pour notre grammaire initiale. Parmi les propriétés qu'elle contient, nous distinguons des parties du discours (verbe, nom, nom propre, pronom, déterminant), dont l'usage sera similaire aux propriétés booléen, entier et opérateur dans l'exemple du chapitre précédent : ces propriétés seront surtout utilisées comme des symboles non-terminaux dans la grammaire d'approximation, restreignant la catégorie syntaxique des feuilles pouvant apparaître à certains endroits de la structure abstraite. D'autres propriétés (transitif, intransitif), n'apparaîtront que dans les contraintes de bonne formation, ou dans le cadre de réalisations conditionnelles lors de la linéarisation, guidant l'usage

ou l'interprétation des phrases qui les contiennent de manière plus subtile. Elles incluront principalement des propriétés morphosyntaxiques ou liées à la sous-catégorisation (valence, restrictions de sélection, *etc.*).

un	déterminant
chien	nom
François	nom propre
il	pronom
regarde	verbe, transitif
marche	verbe, intransitif

TABLE 4.1 – Exemples d'entrées lexicales pour le langage abstrait

Tout au long de ce chapitre, le langage abstrait et son lexique seront progressivement étendus afin d'enrichir la grammaire. Nous introduirons en particulier de nouvelles propriétés lexicales et étiquettes d'arête afin de modéliser les phénomènes nouvellement couverts par la grammaire.

Forme de surface La forme phonologique d'un énoncé, désignée dans la suite par le terme de *forme de surface*, est, formellement, un mot sur un alphabet Σ constitué par l'ensemble des mots du lexique. Le langage concret que nous construirons sera formé de lambda-termes, représentant des formes de surface de la manière évoquée dans la section 2.2.4. Ces lambda-termes s'appuieront sur un ensemble de constantes de type $* \to *$, correspondant aux mots du lexique.

Pour faciliter la lecture de ces termes, nous omettrons systématiquement l'opération de concaténation ainsi que les annotations de typage, pour ne conserver qu'une représentation directe du mot résultant. Ainsi, la concaténation $(.\ a\ (.\ b\ c)) =_\beta \lambda x.a\ (b\ (c\ x))$ de trois termes a, b et c représentant des chaînes sera simplement notée $a\ b\ c$, sans ambigüité supplémentaire en raison de l'associativité de la concaténation.

Sémantique Pour représenter le sens des énoncés, nous nous appuierons tout au long de ce chapitre sur la sémantique par la théorie des modèles initialement proposée par Montague. Dans ce cadre, le sens d'une phrase déclarative est dénoté par une formule logique, et le monde dans lequel cet énoncé se situe est associé à un modèle mathématique. Ainsi, la notion de vérité d'un énoncé coïncide avec la satisfaction de la formule logique représentant son sens, par le modèle correspondant au monde auquel il appartient, aux termes d'une interprétation fixée à l'avance. Nous adopterons plusieurs conventions issues de cette tradition, en particulier en matière de traitement des quantifications par passage de continuation. Nous reviendrons à l'issue de ce chapitre sur le choix de cet outil, notamment dans le but de simplifier nos règles de linéarisation sémantiques.

Concrètement, nos réalisations en sémantique seront des lambda-termes représentant des formules logiques, suivant la méthode présentée dans le dernier paragraphe de la section 2.2.4. Afin de distinguer visuellement ces formules sémantiques de celles employées pour contraindre la grammaire et guider la linéarisation, nous emploierons une police spécifique pour les connecteurs logiques et quantificateurs sémantiques : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \exists, \forall$. En outre, pour faciliter la lecture des lambda-termes sémantiques, nous omettrons l'abstraction lors d'une quantification, en notant par exemple $\exists x.M$ au lieu de $\exists \lambda x.M$; nous emploierons également les connecteurs comme des opérateurs infixes, en leur attribuant leurs priorités usuelles : ainsi, le terme $M \vee \neg N \wedge O$ abrégera le terme $\vee\ M\ (\wedge\ (\neg\ N)\ O)$, par exemple.

Outre les constantes correspondant aux connecteurs logiques et quantificateurs, nos règles de linéarisation sémantiques pourront inclure des constantes représentant les prédicats du lexique (verbes, noms, *etc.*), tandis que les déterminants et pronoms se verront associer des termes d'ordre supérieur assurant la quantification. La nature exacte de ces constantes sera détaillée lors de la constructions de la linéarisation sémantique.

Les types atomiques sur lesquels s'appuient nos lambda-termes sémantiques sont e et t, correspondant respectivement aux éléments (les objets du modèle représentant le monde) et aux propositions (valeurs de vérité des énoncés). Par convention, les abstractions portant sur un type atomique (usuellement e) emploieront pour variable une lettre minuscule $(x, y, z, \dots)$; tandis que celles portant sur des termes d'ordre supérieur (par exemple de type $e \to t$) emploieront une majuscule $(P, Q, \dots)$. Ainsi, nous pourrons omettre les annotations de type de la plupart des termes, leur typage étant implicite par convention (par exemple dans $\lambda P.\lambda Q.\lambda x.P\ x \wedge Q\ x$).

Pour finir, l'approche que nous décrivons ici associe aux énoncés une sémantique extensionnelle : nous avons opté pour ce modèle pour préserver la simplicité de nos exemples sémantiques. Toutefois, linéariser nos structures abstraites vers une représentation sémantique intensionnelle (permettant d'exprimer les notions de croyance et de mondes possibles) ou tout autre modèle demeure également possible, aussi longtemps que les formules correspondantes peuvent être représentées par le biais du lambda-calcul simplement typé.

4.2 Grammaire initiale

Ayant spécifié les types d'objets que nous manipulerons, nous proposons maintenant une grammaire constituant un modèle simple de la phrase en français. Cette grammaire sera par la suite enrichie progressivement au fil des prochaines sections.

En suivant la méthode exposée tout au long du chapitre 3, nous décrivons d'abord le langage abstrait, modélisant la syntaxe profonde, par le biais d'une

grammaire régulière d'approximation, que nous complèterons ensuite par des contraintes logiques de bonne formation. Nous proposerons enfin des règles de linéarisation en surface et en sémantique pour les productions de cette grammaire.

Grammaire d'approximation Notre grammaire support emploiera deux symboles non-terminaux, P et A, qui correspondent respectivement aux propositions et à leurs arguments ; le symbole P tenant lieu d'axiome. Elle se compose de quatre productions, données par la figure 4.1.

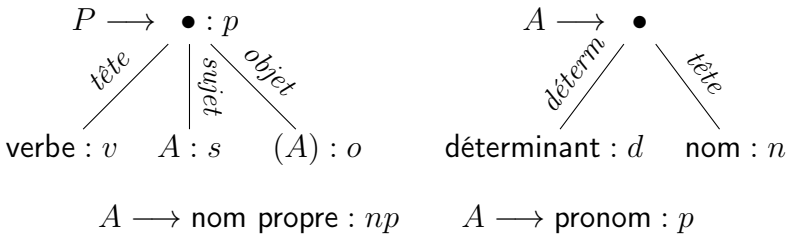


FIGURE 4.1 – Grammaire d'approximation initiale

La première production (en haut à gauche) permet de construire une proposition, dont la tête syntaxique (étiquetée v pour la suite) est une entrée lexicale ayant la propriété verbe. Elle est munie d'un argument sujet (étiqueté s), ainsi que d'un argument objet optionnel (o). Les autres productions permettent de construire un argument comme un syntagme nominal, pouvant être composé d'une paire déterminant/nom (le nom étant dans ce cas considéré comme la tête syntaxique), d'un nom propre, ou bien d'un pronom.

Contraintes logiques Comme suggéré dans le chapitre 3, la grammaire d'approximation précédente, complétée par un lexique tel que celui suggéré par la table 4.1, ne décrit pas exactement le langage des structures abstraites que nous souhaitons considérer comme valides. Elle permet en effet de construire des phrases mal formées, ne tenant pas compte, par exemple, de la transitivité d'un verbe. La figure 4.2 illustre ce problème de sur-génération : les deux structures abstraites qu'elle dépeint sont sanctionnées par la grammaire support, mais celle de droite comprend un usage transitif du verbe **marche**, ce qui ne devrait pas être permis par le lexique.

Nous raffinons maintenant le langage abstrait esquissé par la grammaire d'approximation précédente, par le biais de contraintes utilisant le langage logique décrit section 3.3. Nous nous contenterons initialement de garantir le respect de la transitivité des verbes, en associant dans la première production l'existence d'un argument objet optionnel (o) avec la propriété lexicale de transitivité du verbe qui lui correspond (v). Cette contrainte est traduite par la formule suivante : $\text{avec}(o) \Leftrightarrow \mathsf{transitif}(v)$.

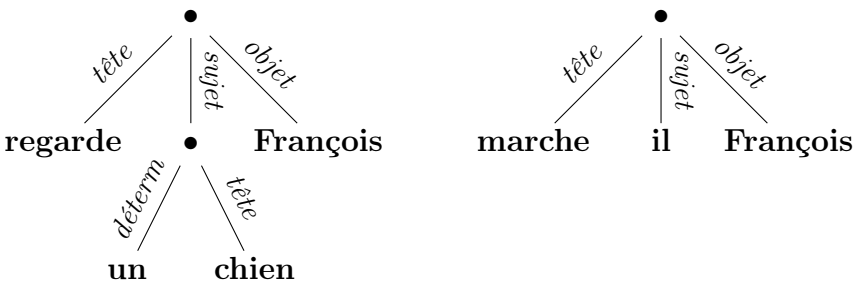


FIGURE 4.2 – Deux structures abstraites non contraintes

La structure de droite de la figure 4.2 est ainsi correctement exclue, tout en préservant de la validité de la structure de gauche. Cette dernière sera utilisée dans la suite pour exemplifier le résultat des linéarisations, en surface et en sémantique.

Linéarisations Nous donnons maintenant, pour chacune des productions de la grammaire, des règles de linéarisation construisant la forme de surface des énoncés, ainsi que leur sémantique.

Forme du surface Pour l'instant, les règles de linéarisation produisant la forme de surface demeurent très simples ; elles ne contiennent pas de réalisations multiples ou de conditions. Nous rappelons les quatre productions de notre grammaire, en précisant à droite de chaque production la règle de linéarisation qui lui est associée :

1. $P \longrightarrow \bullet : p$ (tête, sujet, objet) verbe : v ; $A : s$; $(A) : o$ — $s\ v\ o$
2. $A \longrightarrow \bullet$ (déterm, tête) déterminant : d ; nom : n — $d\ n$
3. $A \longrightarrow$ nom propre : np — np
4. $A \longrightarrow$ pronom : p — p

Cette linéarisation produit simplement le résultat de la concaténation des entrées lexicales de la phrase en suivant l'ordre des mots usuel en français : la première règle construit une proposition suivant l'ordre SVO, la seconde place le déterminant avant le nom, et les deux dernières reprennent simplement la forme de surface de l'entrée lexicale qui réécrit le symbole A.

Ainsi, dans le cas de la structure abstraite valide donnée par la figure 4.2 (à gauche), la réalisation associée est la séquence de mots attendue : « un

chien regarde François ». La seconde production (instanciée pour construire l'argument *sujet*) combine les mots associées aux entrées lexicales de ses variables d et n pour produire la chaîne **un chien**, et la troisième production produit l'argument *objet*, en recopiant la réalisation de son unique variable np (ici, **François**) ; enfin, la première production concatène les réalisations de ses trois arguments s, v et o (dans cet ordre) pour produire la réalisation associée à la phrase.

Remarquons que la variable o dans la première règle correspond à un nœud optionnel : nous considérons ici par simplicité que le lambda-terme associé aux feuilles étiquetées $\perp$ est la constante ε dénotant le mot vide. Une manière plus explicite d'aboutir au même résultat pourrait être d'employer une linéarisation conditionnelle comme suit :

1. $P \longrightarrow \bullet : p$; tête : verbe : v ; sujet : $A : s$; objet : $(A) : o$

$$\left\{ \begin{array}{lcl} \text{avec}(o) & \longrightarrow & s\ v\ o \\ \text{sans}(o) & \longrightarrow & s\ v \end{array} \right\}$$

Les résultats produits par ces deux variantes de la première règle de linéarisation sont identiques. Par la suite, nous adopterons fréquemment pour des raisons pratiques cette convention d'attribuer une valeur « par défaut » aux arguments optionnels absents, sans spécifier systématiquement des linéarisations alternatives pour chaque cas.

Sémantique Nous décrivons maintenant la linéarisation sémantique de nos structures abstraites, en commençant par détailler les réalisations associées à leurs entrées lexicales.

Chaque entrée du lexique est associée à un lambda-terme sur la signature logique évoquée précédemment (*cf.* page 73). Dans la plupart des cas, ce lambda-terme se réduit à une constante, dont le type dépend de la catégorie syntaxique du mot en question ; d'autres entrées (en particulier des grammèmes comme les déterminants) auront pour réalisations des lambda-termes complexes. La table 4.2 donne quelques exemples de réalisations associées à des entrées lexicales.

François	$\boldsymbol{François}^{e}$	**regarde**	$\boldsymbol{regarde}^{e \to e \to t}$
chien	$\boldsymbol{chien}^{e \to t}$	**un**	$\lambda P.\exists x.P\ x$
marche	$\boldsymbol{marche}^{e \to t}$		

TABLE 4.2 – Lambda-termes sémantiques associés aux entrées lexicales

Les noms propres sont ainsi simplement associés à des constantes de type e, correspondant à l'entité qu'ils dénotent. Les constantes associées aux noms communs sont des prédicats de type $e \to t$, le terme « **livre** x » s'interprétant comme vrai lorsque l'élément désigné par x est un livre dans le modèle. Le

type des constantes associées au verbe dépend de leur valence : ainsi, une constante associée à un verbe intransitif (comme **marche**) a le type $e \to t$, tandis que celle associée à un verbe transitif (comme **regarde**) requiert un argument supplémentaire, ayant le type $e \to e \to t$. Leur interprétation sera similaire à celle des noms communs, vérifiant que l'action dénotée par le verbe est bien effectuée par ses arguments d'après le modèle.

D'autres entrées lexicales, comme les déterminants, se voient associer un terme plus complexe, ayant le même type qu'un quantificateur, dont ils ont le rôle. Ainsi, la réalisation sémantique du déterminant **un** est $\lambda P.\exists x.P\ x$, affirmant l'existence d'un élément x, qui aura par la suite les propriétés spécifiées par le syntagme nominal et la proposition auquel il appartient. Cette conjonction de propriétés est désignée par la variable (abstraite) P, dont la valeur sera construite progressivement lors de la linéarisation de la phrase en sémantique. Cette interprétation des déterminants suit l'analyse proposée par Montague [1988].

Par suite, les règles de linéarisation en sémantique attachées aux productions feront parfois usage de constantes supplémentaires, notées $\mathbf{\Omega}$. Ces constantes représentent un contenu sémantique vide du type correspondant : $\mathbf{\Omega}^e$ ou $\mathbf{\Omega}^{e\to t}$ ont respectivement le type d'un élément ou d'une propriété, mais n'ont pas d'interprétation sémantique associée. Nous en ferons ici usage dans la linéarisation de l'objet optionnel o de la première production : en l'absence d'un tel objet, la réalisation associée par défaut à sa variable o sera $\lambda P.P\ \mathbf{\Omega}^e$. La présence de cette constante permet de construire un terme bien typé lors de la linéarisation d'une proposition intransitive, et la constante $\mathbf{\Omega}^e$ associée à l'objet du verbe sera ensuite effacée par β-réduction du lambda-terme associé à la production. Ce terme vide de contenu, non apparent dans la réalisation finale, peut être comparée à l'emploi de la chaîne vide ε dans la linéarisation vers la forme de surface.

Nous pouvons maintenant donner les règles de linéarisation associées à chacune des productions de la grammaire :

1. $P \longrightarrow \bullet : p$ (*tête* : verbe : v ; *sujet* : $A : s$; *objet* : $(A) : o$)
   $$s\ \lambda x.o\ \lambda y.\begin{Bmatrix}\mathsf{transitif}(v) & \longrightarrow v\ x\ y\\ sinon & \longrightarrow v\ x\end{Bmatrix}$$
2. $A \longrightarrow \bullet$ (*déterm* : déterminant : d ; *tête* : nom : n)
   $$\lambda P.d\ \lambda x.n\ x \wedge P\ x$$
3. $A \longrightarrow \mathsf{nom\ propre} : np$ — $\lambda P.P\ np$
4. $A \longrightarrow \mathsf{pronom} : p$ — p

Afin de permettre le traitement de la quantification, la règle de linéarisation associée à la première production abstrait l'objet éventuel du verbe, et applique au terme résultant le terme associé à son objet, puis procède de même pour son sujet. Cette règle fait usage du langage de macros détaillé dans la section 3.5.1, pour factoriser le mode d'application du sujet et de l'objet, indépendamment de la transitivité du verbe.

Subsidiairement, remarquons que ce choix de construction n'est pas entièrement innocent. Considérons la règle de linéarisation alternative suivante :

1. $P \longrightarrow \bullet : p$; $\mathsf{verbe} : v$ (tête), $A : s$ (sujet), $(A) : o$ (objet)

$$s\ \lambda x. \left\{ \begin{array}{cc} \mathsf{transitif}(v) & \longrightarrow o\ \lambda y. v\ x\ y \\ sinon & \longrightarrow v\ x \end{array} \right\}$$

Fonctionnellement, cette règle est équivalente à la précédente. Elle serait cependant moins aisée à enrichir si nous souhaitions, par exemple, tenir compte des ambigüités de portée dans la quantification, en permettant également de placer la quantification de l'objet en amont de celle du sujet. À titre d'exemple, le choix de quantifier sur le sujet d'abord et sur l'objet ensuite force l'interprétation d'une phrase telle que « Un service secret a conçu chaque méthode de chiffrement connue. » par « Un unique service secret est à l'origine de toutes les méthodes de chiffrement connues. » ; l'autre lecture possible de cette phrase (« Toute méthode de chiffrement connue peut être associée au service secret particulier qui l'a mise au point. »), qui constitue une affirmation plus faible, est exclue par la règle de linéarisation naïve donnée plus haut.

Afin de permettre les deux lectures, nous pourrions proposer une règle de linéarisation factorisant le noyau verbal dans la première version de notre règle par une variable nv, et produisant indifféremment les deux modes de quantification. Cette règle est plus éloignée dans sa structure de la version alternative que nous avons écartée, et se formule comme suit :

1. $P \longrightarrow \bullet : p$; $\mathsf{verbe} : v$ (tête), $A : s$ (sujet), $(A) : o$ (objet)

$$\left\{ \begin{array}{c} s\ \lambda x. o\ \lambda y. nv \\ o\ \lambda y. s\ \lambda x. nv \end{array} \right\}$$

$$où\ nv = \left\{ \begin{array}{cc} \mathsf{transitif}(v) & \longrightarrow v\ x\ y \\ sinon & \longrightarrow v\ x \end{array} \right\}$$

Retournant aux règles de linéarisation que nous avons choisi de proposer, la seconde règle construit le sens d'un groupe nominal en appliquant la quantification apportée par son déterminant (d) au terme $\lambda x. n\ x \wedge P\ x$, qui vérifie que l'élément sur lequel porte la quantification (identifié avec x par β-réduction) satisfait bien la propriété donnée par son nom commun $(n\ x)$, ainsi que la propriété supplémentaire P – où P est une propriété abstraite, qui sera déterminée par le contexte du groupe nominal ; suivant l'interprétation usuelle.

Les deux dernières règles de linéarisation construisent également des formules de type $(e \to t) \to t$, soit en appliquant directement la propriété abstraite P donnée par le contexte à la constante correspondant à l'entité nommée en propre, soit en employant directement la réalisation associée à un pronom. Nous ne détaillons pas ici cette réalisation (complexe), qui doit tenir compte du contexte de l'énoncé et traiter le problème de l'anaphore.

Ainsi, nous pouvons détailler la réalisation sémantique de la structure valide donnée par la figure 4.2 de manière ascendante. Le symbole non-terminal correspondant au groupe nominal sujet se voit associer, selon la seconde règle de linéarisation, le terme $\lambda P.\exists x.\textbf{\textit{chien}}\ x \wedge P\ x$ (après β-réduction). Le groupe nominal objet est, pour sa part, réalisé comme $\lambda P.P\ \textbf{\textit{François}}$. Pour obtenir la réalisation associée à l'ensemble de la structure, nous remplaçons respectivement dans la règle de linéarisation associée à la première production les variables s, o et v par les réalisations associées aux arguments *sujet* et *objet* et par celle associée à l'entrée lexicale du verbe ($\textbf{\textit{regarde}}^{e\to e\to t}$). Le verbe **regarde** ayant la propriété transitif, le lambda-terme résultant est le suivant :

$$(\lambda P.\exists x.\textbf{\textit{chien}}\ x \wedge P\ x)\ \lambda x.(\lambda P.P\ \textbf{\textit{François}})\ \lambda y.\textbf{\textit{regarde}}\ x\ y$$

Équivalant, par β-réduction, à : $\exists x.\textbf{\textit{chien}}\ x \wedge \textbf{\textit{regarde}}\ x\ \textbf{\textit{François}}$.

4.3 Traitement de l'accord

Nous souhaitons maintenant enrichir progressivement cette grammaire, en commençant par inclure un traitement de l'accord en genre, nombre et personne. Ceci nous permettra de restreindre davantage le langage résultant, en garantissant l'accord dans une proposition (sujet/verbe) ainsi qu'au sein d'un groupe nominal (déterminant/nom).

Cette extension ne requiert pas de modifier la structure générale des productions de la grammaire : nous ajoutons seulement au lexique un ensemble de propriétés traduisant le genre, le nombre et la personne associés à certaines entrées, et complétons l'ensemble des contraintes logiques de la grammaire pour forcer l'accord entre les entrées d'un même syntagme. Le processus de linéarisation, pour sa part, demeure inchangé.

Alternativement, nous aurions pu traiter l'accord de manière asymétrique, en attribuant des propriétés lexicales d'accord uniquement aux noms et pronoms, et en incluant dans la linéarisation des règles morphologiques permettant de former, par exemple, le pluriel d'un déterminant ou d'un verbe. Les langages concrets issus de ces deux approches seraient identiques, en surface comme en sémantique. Nous avons ici opté pour l'emploi d'une linéarisation directe, sans règles morphologiques.

Propriétés lexicales Nous complétons nos entrées lexicales à l'aide d'un ensemble de propriétés caractérisant le genre (masculin, féminin), le nombre

(singulier, pluriel) et la personne (1re personne, 2e personne, 3e personne). Dans le cas où un mot du lexique appartient à une catégorie syntaxique portant une marque d'accord, mais que sa forme est ambigüe, plusieurs entrées lexicales lui sont associées. Chacune de celles-ci correspond à l'une des interprétation possibles du mot, et elles ne diffèrent que par leurs listes de propriétés. Quelques exemples d'entrées lexicales mises à jour avec ces informations sont données par la table 4.3.

il	pronom, 3e personne, masculin, singulier
elles	pronom, 3e personne, féminin, pluriel
un	déterminant, masculin, singulier
chien	nom, masculin, singulier
souris	nom, féminin, singulier
souris	nom, féminin, pluriel
François	nom propre, masculin, singulier
regarde	verbe, 1re personne, singulier
regarde	verbe, 3e personne, singulier

TABLE 4.3 – Extrait de lexique incluant des propriétés d'accord

Remarquons que les entrées lexicales multiples (telles que **souris** ou **regarde**) sont distinctes dans les structures profondes, mais que les linéarisations en surface se restreignent au mot tel qu'il apparaît dans la colonne de gauche. Ainsi, le mot « souris » tel qu'il apparaît dans la réalisation d'une phrase n'est pas marqué en nombre, et correspond à deux structures abstraites distinctes (l'une au singulier et l'autre au pluriel) dont l'une est sans doute mal formée.

L'emploi d'entrées lexicales multiples dans ces cas permet de simplifier les contraintes logiques liées à l'accord, par rapport à l'emploi d'une unique entrée lexicale sous-spécifiée. Supposons en effet que l'entrée lexicale **souris** ait été sous-spécifiée, et soit compatible avec des déterminants ou adjectifs singuliers ou pluriels : dans ce cas, la validité d'une structure abstraite dans une grammaire plus riche (incluant, par exemple, des adjectifs) aurait dépendu des relations entre tous les composants d'un syntagme nominal. Par exemple, la structure abstraite décrite par la séquence de feuilles « **les souris verte** » est invalide ; mais ni le déterminant, ni l'adjectif ne sont incompatibles avec la tête, puisque le nombre de celle-ci est sous-spécifié : il conviendrait alors de comparer les marques d'accord entre tous les éléments du syntagme, requérant des contraintes plus élaborées que celles que nous donnons dans la suite (mais néanmoins formulables dans notre langage logique). La forme des entrées apparaissant dans notre lexique actuel permet d'éviter ces problèmes.

Prédicats et relations additionnels L'un des principaux intérêts de l'emploi d'un langage logique pour la contrainte grammaticale est son extensibilité :

nous pouvons combiner les primitives à notre disposition (opérateurs, quantificateurs, prédicats liés aux propriétés lexicales et relations de domination) pour construire des prédicats et relations plus riches, qui traduisent directement des concepts linguistiques de plus en plus complexes. Comme évoqué dans le chapitre 3, nous employons l'opérateur $\triangleq$ pour définir un nouveau prédicat (à gauche) à partir d'une formule complexe (à droite).

Ainsi nous pouvons, par exemple, construire une relation qui traduit le respect des contraintes de sous-catégorisation imposées par notre grammaire :

$$\begin{aligned} \textit{sous-catégorisation}(\textit{verbe}, \textit{sujet}, \textit{objet}) \triangleq\ & \text{avec}(\textit{objet}) \Rightarrow \mathsf{transitif}(\textit{verbe}) \\ & \wedge\ \text{sans}(\textit{objet}) \Rightarrow \mathsf{intransitif}(\textit{verbe}) \end{aligned}$$

La définition de cette relation peut être envisagée comme la mise en œuvre (implémentation) du concept linguistique de sous-catégorisation d'un verbe dans notre grammaire, sous la forme d'une relation entre trois nœuds *verbe*, *sujet* et *objet*. L'objectif est que, par la suite, toute amélioration du traitement des contraintes de sous-catégorisation dans notre grammaire passe par un changement ponctuel dans cette définition, plutôt qu'une revue exhaustive des contraintes sur les productions liées aux syntagmes verbaux.

Nous introduisons maintenant une série de prédicats liés à l'accord, permettant de parler du genre, du nombre et de la personne d'un syntagme. Un syntagme possède ainsi le genre et le nombre de sa tête. Celle-ci peut-être localisée par l'expression régulière $\textit{tête}^*$ sur les chemins présents dans la structure abstraite : une entrée lexicale est ainsi sa propre tête, ainsi que celle de tous les syntagmes la dominant par l'intermédiaire d'une séquence d'arêtes étiquetées *tête*. Cette dernière catégorie se limite pour l'instant aux paires déterminant/nom produites par la seconde production, mais comprendra par la suite des syntagmes nominaux plus riches, incluant des modificateurs. Formellement, les définitions ainsi introduites s'appuient sur des relations construites à partir d'expressions régulières (voir définition 3.3), et se formulent comme suit :

$$\begin{aligned} \textit{g\_masculin}(x) &\triangleq \exists t.\mathsf{masculin}(t) \wedge \textit{tête}^*(x,t) \\ \textit{g\_féminin}(x) &\triangleq \exists t.\mathsf{féminin}(t) \wedge \textit{tête}^*(x,t) \\ \textit{n\_singulier}(x) &\triangleq \exists t.\mathsf{singulier}(t) \wedge \textit{tête}^*(x,t) \\ \textit{n\_pluriel}(x) &\triangleq \exists t.\mathsf{pluriel}(t) \wedge \textit{tête}^*(x,t) \end{aligned}$$

Par ailleurs, un syntagme est considéré comme étant de la troisième personne si et seulement si il ne se réduit pas à une entrée lexicale à la première ou deuxième personne ; cette dernière condition repose sur l'hypothèse qu'un pronom est le seul élément de son syntagme nominal (il ne peut pas être déterminé ou modifié) ; nous abandonnons pour cette raison l'emploi de l'expression $\textit{tête}^*$. Par suite, les syntagmes construits autour d'un nom seront systématiquement traités comme étant de la troisième personne du point de vue de l'accord. Réciproquement, aucune définition logique supplémentaire n'est requise pour

caractériser un syntagme à la première ou deuxième personne : les propriétés lexicales 1^{re} personne et 2^{e} personne des pronoms suffisent à cette tâche. La formule correspondant à la notion de troisième personne est donc donnée par :

$$3^{e}\ personne(x) \triangleq \neg(1^{re}\ \mathsf{personne}(x) \vee 2^{e}\ \mathsf{personne}(x))$$

Nous pouvons maintenant définir simplement les relations traduisant l'accord en genre, en nombre et en personne :

$$\begin{aligned}
accord_genre(x,y) &\triangleq g_masculin(x) \wedge g_masculin(y)\\
&\vee\ g_féminin(x) \wedge g_féminin(y)\\
accord_nombre(x,y) &\triangleq n_singulier(x) \wedge n_singulier(y)\\
&\vee\ n_pluriel(x) \wedge n_pluriel(y)\\
accord_personne(x,y) &\triangleq 1^{re}\ \mathsf{personne}(x) \wedge 1^{re}\ \mathsf{personne}(y)\\
&\vee\ 2^{e}\ \mathsf{personne}(x) \wedge 2^{e}\ \mathsf{personne}(y)\\
&\vee\ 3^{e}\ personne(x) \wedge 3^{e}\ personne(y)
\end{aligned}$$

Mise à jour des contraintes logiques En nous appuyant sur les définitions précédentes, nous pouvons maintenant mettre à jour les contraintes de la grammaire pour tenir compte de l'accord. Pour rappel, la figure 4.1 résume les productions de la grammaire actuelle.

Nous remplaçons la contrainte précédente sur la première production (portant sur la transitivité) par la formule atomique $sous\text{-}catégorisation(v, s, o)$. La grammaire résultante est exactement équivalente en termes de langage généré. La seule différence réside dans le fait que la contrainte modifiée traduit maintenant explicitement son utilité : elle représente exactement les contraintes de sous-catégorisation mises en œuvre dans la grammaire, ce qui permettra de faciliter de futures extensions.

Par suite, nous ajoutons deux contraintes liées à l'accord. La première s'applique à la première production, et garantit l'accord en nombre et en personne du verbe avec son sujet : $accord\_nombre(s, v) \wedge accord\_personne(s, v)$. La seconde s'applique à la deuxième production, qui construit un argument comme une paire déterminant/nom, et garantit l'accord en genre et en nombre entre ces deux éléments : $accord\_genre(d, n) \wedge accord\_nombre(d, n)$.

Le langage abstrait résultant de l'ajout de ces contraintes est plus restreint que celui de la grammaire initiale, éliminant nombre de structures mal formées. L'exclusion de ces structures du langage abstrait entraîne celle de leurs linéarisations dans le langage concret, améliorant la précision de notre grammaire.

4.4 Traitement des subordonnées

Nous souhaitons maintenant inclure dans notre grammaire un traitement des propositions subordonnées. Ce traitement inclura des propositions complé-

tives sujet et objet, ainsi que la modification d'un syntagme nominal par une proposition relative, en tenant compte de l'accord entre antécédent et pronom relatif. Un pronom relatif peut aussi bien tenir lieu de sujet que d'objet dans sa proposition [1] ; la linéarisation en surface devra donc permettre l'antéposition du pronom par rapport à la relative. Enfin, un pronom relatif objet peut apparaître dans une proposition subordonnée à la relative à laquelle il est lié ; cette relation de subordination pouvant être indirecte, de sorte que plusieurs complétives successives séparent l'antécédent du pronom relatif. Ce dernier cas de figure n'est possible que dans certaines configurations, dépendant des contraintes dites d'îlot étudiées par Ross [1967] ; et entraîne des dépendances syntaxiques entre des composants arbitrairement lointains, à la fois dans la réalisation de la phrase et dans sa représentation abstraite. Ces dernières contraintes sont qualifiées de dépendances à longue distance, et font l'objet de traitements très divers dans la littérature (*e.g.* Chomsky [1977], Kaplan et Zaenen [1995], ...). Nous présenterons dans cette section une approche rappelant la seconde, proposée dans le cadre des grammaires lexicales-fonctionnelles.

Concrètement, un exemple de phrase que nous souhaitons pouvoir traiter est : « Le livre que je tiens affirme que notre monde est plat. » ; la structure abstraite associée est donnée par la figure 4.3. L'argument objet du verbe « affirme » dans la proposition principale y est réécrit comme une proposition subordonnée complétive ; et son sujet est, comme précédemment, un groupe nominal. Ce dernier est modifié par une proposition subordonnée relative, dans laquelle l'un des arguments est un pronom relatif. Remarquons que l'ajout de modificateurs aux groupes nominaux requiert l'introduction d'une nouvelle étiquette d'arête dans notre langage abstrait (*mod*).

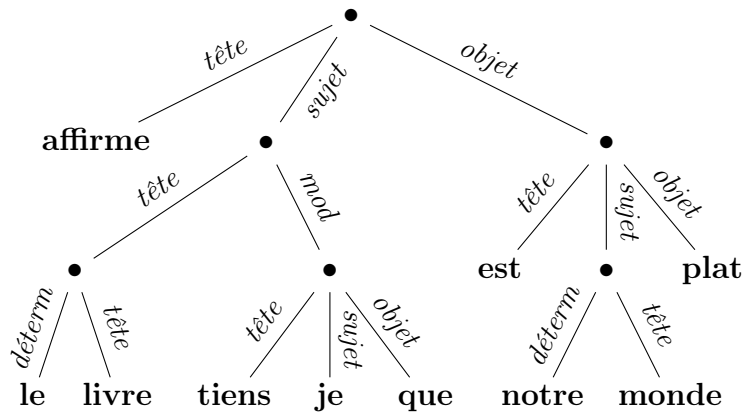


FIGURE 4.3 – Exemple de structure abstraite contenant des subordonnées

1. Par exemple, le pronom relatif **qui**/**que** est sujet du verbe **lire** dans « le livre qui est lu », alors qu'il est son objet dans « le livre que Paul lit ».

Structure grammaticale Afin de permettre la construction de structures abstraites semblables à celle proposée figure 4.3, nous ajoutons maintenant deux autres productions aux quatre de la grammaire d'approximation initiale. Les règles ainsi introduites sont données par la figure 4.4 ; nous nous référerons à elles dans la suite comme à la cinquième et sixième production de la grammaire support.

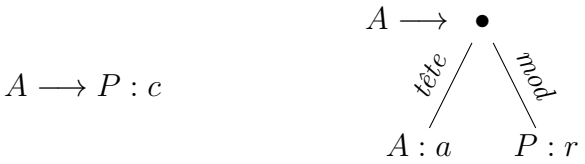


FIGURE 4.4 – Productions supplémentaires pour l'ajout de subordonnées

La cinquième production, à gauche, permet de réécrire un argument verbal (A) comme une proposition (P) ; elle permet l'inclusion de propositions complétives (sujet et objet) dans la grammaire (« que notre monde est plat »). Observons que cette production rend notre grammaire récursive, en lui permettant d'imbriquer plusieurs complétives (subordonnées les unes aux autres). La sixième production, à droite, permet d'ajouter un modificateur à un argument, sous la forme d'une proposition ; cette règle traduit l'ajout de propositions relatives (« que je tiens ») qualifiant le reste du groupe nominal (« le livre »). Dans le syntagme nominal résultant, le nœud dominé par l'arête *tête* correspond à l'antécédent, et celui dominé par *mod* à la relative.

Une fois encore, l'ensemble des structures abstraites générées par cette grammaire contient de nombreux éléments invalides requérant d'être filtrés. Nous décrirons dans la suite les contraintes de sous-catégorisation associées aux complétives et celles portant sur le placement des pronoms relatifs. Celles-ci devront cependant s'appuyer sur des prédicats et relations nouvellement définis, ainsi que sur des propriétés lexicales additionnelles que nous introduisons maintenant.

Propriétés lexicales Les propriétés supplémentaires venant enrichir notre lexique sont relatif, nominatif, accusatif, complétive objet, objet nominal, complétive sujet et sujet nominal. Les premières d'entre elles concernent les pronoms : la propriété relatif dénote un pronom relatif, et les propriétés nominatif et accusatif dénotent les fonctions syntaxiques qu'il peut occuper (respectivement sujet et objet, héritage des cas morphologiques). Les autres propriétés apportent des informations sur la sous-catégorisation des verbes : complétive objet, objet nominal, complétive sujet et sujet nominal dénotent respectivement qu'un verbe peut avoir pour objet une proposition complétive ou un syntagme nominal, et de même pour son sujet. Des exemples d'entrées lexicales illustrant l'usage de ces propriétés sont fournies par les Tables 4.4 et 4.5.

que	pronom, relatif, accusatif, masculin, singulier
que	pronom, relatif, accusatif, féminin, singulier
que	pronom, relatif, accusatif, masculin, pluriel
que	pronom, relatif, accusatif, féminin, pluriel
qui	pronom, relatif, nominatif, masculin, singulier
qui	...
lequel	pronom, relatif, nominatif, masculin, singulier
lequel	pronom, relatif, accusatif, masculin, singulier
laquelle	pronom, relatif, nominatif, féminin, singulier
laquelle	...
lesquels	...
lesquelles	...

TABLE 4.4 – Entrées lexicales décrivant des pronoms relatifs

Remarquons que certains pronoms relatifs (**qui**, **que**) ne portent pas de marque d'accord en genre et en nombre, tandis que d'autres (**le/la/lesquel-le-s**) sont partiellement indifférents à la fonction syntaxique dans laquelle ils sont utilisées (sujet ou objet direct). Le lexique sur lequel nous nous appuyons développe toute la combinatoire associée à ces pronoms relatifs. Le développement d'une grammaire à large couverture requerrait l'emploi d'une représentation compacte pour ces multiples entrées. Le cas échéant, il serait également possible d'intégrer des règles morphologiques à la linéarisation en surface des phrases, comme évoqué précédemment. Ceci permettrait de construire la forme appropriée pour chaque pronom relatif dynamiquement au moment de la réalisation, par opposition au lexique statique, pré-calculé, dont nous disposons actuellement.

affirme	verbe, 3^{e} personne, singulier, transitif, sujet nominal, complétive objet, objet nominal
regarde$_1$	verbe, 3^{e} personne, singulier, transitif, sujet nominal, objet nominal
regarde$_2$	verbe, 3^{e} personne, singulier, transitif, complétive sujet, sujet nominal, objet nominal

TABLE 4.5 – Entrées lexicales décrivant des verbes acceptant des complétives

Nous distinguons en outre les différents usages d'un verbe, afin de permettre un traitement valide en sémantique. Le verbe « regarder » possède ainsi deux sens : le premier comme synonyme de voir, requérant un sujet nominal ; et le second correspondant à l'emploi familier comme synonyme de concerner, qui admet plusieurs usages (« La date de mon départ me regarde. » ou « Que je décide de partir ou non me regarde. »).

Maintenant que nous avons enrichi notre collection de propriétés lexicales,

nous devons également étendre notre vocabulaire logique pour pouvoir formuler les contraintes de grammaticalité portant sur les relatives. Dans ce but, nous explicitons tout d'abord la nature de ces contraintes linguistiques, avant de décrire la mise en œuvre des notions associées dans notre vocabulaire logique.

Fonctions syntaxiques des pronoms relatifs La formation d'une relative obéit à un ensemble de contraintes précis : le pronom relatif antéposé peut avoir fonction de sujet ou d'objet dans la proposition, mais il peut également correspondre à l'argument d'une proposition complétive qui lui est subordonnée. Cette possibilité de relier une relative à son pronom par le biais d'un chemin complexe (dit *chemin wh*, en raison de la forme des pronoms relatifs et interrogatifs anglais) fait toutefois l'objet de restrictions importantes, connues sous le nom de *contraintes d'îlot* (*island constraints*). Considérons ainsi les exemples suivants :

1 Le camion qui accélère ...

2 Les livres que tu collectionnes ...

3 Les gens que Luc affirme que je fréquente ...

3a Les gens que Marie pense que Luc affirme que je fréquente ...

3b Les gens que tu dis que Marie pense que Luc affirme que je fréquente

4 Luc affirme que le camion accélère.

4a *Le camion qui Luc affirme qu'accélère ...

5 Que des enfants lisent ce livre inquiète leurs parents.

5a *Ce livre, lequel que des enfants lisent inquiète leurs parents ...

6 L'homme qui a vu l'homme qui a vu l'ours ...

6a L'ours que l'homme qui a vu l'homme qui a vu ...

Les exemples 1 et 2 sont des groupes nominaux modifiés par une relative, dans laquelle le pronom relatif a fonction de sujet et d'objet respectivement. L'exemple 3 illustre la possibilité pour un pronom relatif d'occuper une fonction d'objet dans une complétive subordonnée à la proposition relative à laquelle il correspond ; les exemples 3a et 3b montrant que ce pouvoir n'est pas limité par la « profondeur » de la complétive : le pronom relatif peut être l'objet d'une complétive subordonnée à une complétive subordonnée à la relative, et ainsi de suite.

En revanche, les exemples 4 à 6 soulignent trois limitations imposées par les contraintes d'îlot : un pronom relatif ne peut correspondre au sujet d'une complétive objet (4a), ni à l'argument d'une complétive sujet (5a) ; pas plus qu'il ne peut provenir d'une autre relative enchâssée. L'ensemble des contraintes d'îlot conduisant au rejet de ces énoncés a été, comme mentionné précédemment, implémenté avec concision dans la tradition des grammaires lexicales-fonctionnelles. L'analyse retenue est basée sur la notion d'*incertitude fonctionnelle* (*functional uncertainty*), qui permet au pronom d'être relié à son

antécédent par un chemin régulier dans une structure arborescente (la structure fonctionnelle, ou f-structure) qui représente l'énoncé à un niveau abstrait, et présente de nombreuses similarités avec notre notion de structure abstraite. Aussi, ces contraintes de régularité sur le chemin traversé peuvent se traduire directement dans notre formalisme : par une expression régulière sur les étiquettes d'arêtes, reliant le pronom relatif à la proposition qui lui correspond. L'expression $sujet \mid objet^*$ désigne ainsi soit le sujet d'une relative ; soit son objet, ou l'objet de sa subordonnée complétive objet, ou ainsi de suite. Cette expression recouvre les rôles syntaxiques autorisés pour un pronom relatif, incluant les relatives des exemples 1 à 3 tout en excluant celles des exemples 4 et 5.

Observons enfin que les structures que nous cherchons à exclure de la grammaire ne sont pas exactement représentées par les phrases d'exemple ci-dessus : ces dernières sont au plus une réalisation plausible des structures abstraites que nous souhaitons rejeter. Il est cependant plus simple de raisonner sur ces exemples que sur les structures abstraites qui les sous-tendent. En d'autres termes, nos contraintes filtrent seulement des structures abstraites jugées incorrectes : l'élimination des énoncés agrammaticaux (comme 4a et 5a) est une conséquence du rejet de certaines structures, dont les réalisations en surface auraient été ces énoncés jugés incorrects.

Vocabulaire logique Ayant établi la nature des contraintes que nous souhaitons imposer à nos structures, nous pouvons inclure les notions intermédiaires utiles à leur formulation dans notre vocabulaire logique.

Tout d'abord, nous souhaitons pouvoir identifier par un prédicat une proposition et un syntagme nominal dans une structure abstraite. Selon notre grammaire support, une proposition est une structure dont la tête syntaxique (à distance 1) est un verbe. Un syntagme nominal a pour sa part un nom commun, un nom propre ou un pronom comme tête syntaxique, située à une distance arbitraire dans la structure (puisqu'un syntagme peut se réduire à un pronom, où être composé d'un nom commun accompagné d'un déterminant et de modificateurs). Nous pouvons utiliser à cet effet les relations basées sur les expressions régulières, l'expression $tête^*$ caractérisant la notion de tête syntaxique. La définition des prédicats traduisant ces deux notions dans notre grammaire est alors transparente :

$$\begin{aligned} proposition(x) &\triangleq \exists v.\mathsf{verbe}(v) \wedge tête(x,v) \\ synt\_nominal(x) &\triangleq \exists n.(\mathsf{nom}(n) \vee \mathsf{nom\ propre}(n) \vee \mathsf{pronom}(n)) \wedge tête^*(x,n) \end{aligned}$$

Nous formulons également une nouvelle relation, reliant une proposition relative à toute position que son pronom associé est susceptible d'occuper, en respectant les contraintes d'îlot présentées plus haut :

$$chemin\_wh(x,y) \triangleq sujet(x,y) \vee objet^+(x,y)$$

Afin de réaliser l'antéposition du pronom relatif lors de la linéarisation en surface, et d'assurer la concordance entre une proposition relative et son pronom associé, nous définissons deux prédicats liés à la notion d'extraction. D'une part, un élément est dit extrait si et seulement si il s'agit d'un pronom relatif (ce qui est vrai dans le cadre de notre grammaire actuelle – ce prédicat pourrait être enrichi par la suite pour inclure, par exemple, les pronoms interrogatifs). D'autre part, nous caractérisons l'ensemble des propositions contenant une extraction, comme l'ensemble des propositions contenant un élément extrait au terme d'un chemin d'extraction valide.

$$\begin{aligned} \mathit{extrait}(x) &\triangleq \mathsf{pronom}(x) \wedge \mathsf{relatif}(x) \\ \mathit{prop\_ext}(x) &\triangleq \mathit{proposition}(x) \wedge \exists p.\mathit{extrait}(p) \wedge \mathit{chemin\_wh}(x,p) \end{aligned}$$

Remarquons que, dans la définition du prédicat *prop_ ext*, la première clause (*proposition*(x)) est en fait redondante (compte tenu de notre grammaire support, le nœud x correspond nécessairement à une proposition en raison de l'existence d'un chemin *wh* de x à p). Elle traduit néanmoins explicitement le fait qu'une « proposition contenant une extraction » est, implicitement, une proposition : une éventuelle redéfinition de la nature d'un chemin *wh* (par exemple, à la lumière de nouvelles observations linguistiques) est ainsi moins susceptible de mettre en péril la cohérence de cette définition.

Nous pouvons maintenant nous appuyer sur ces notions pour étendre notre modélisation logique. Nous définissons ainsi une relative comme une proposition contenant un pronom relatif extrait et modifiant un syntagme nominal :

$$\mathit{relative}(x) \triangleq \mathit{prop\_ext}(x) \wedge \exists s.\mathit{synt\_nominal}(s) \wedge \mathit{mod}(s,x)$$

Pour conclure, nous pouvons désormais définir dans notre langage logique la relation d'antécédence : l'antécédent x d'un pronom relatif y est un groupe nominal ; et tous deux sont reliés par l'intermédiaire d'une proposition relative qui modifie l'antécédent, et qui contient le pronom relatif (au terme d'un chemin *wh* valide). La définition est la suivante (pour rappel, l'opérateur $\uparrow$ est issu de la définition 3.3) :

$$\begin{aligned} \mathit{antécédent}(x,y) \triangleq\ & \mathit{synt\_nominal}(x) \wedge \mathsf{pronom}(y) \wedge \mathsf{relatif}(y) \\ & \wedge\ \exists r.\mathit{relative}(r) \wedge \mathit{tête}^* \uparrow \mathit{mod}(x,r) \wedge \mathit{chemin\_wh}(r,y) \end{aligned}$$

La dernière définition complétant notre vocabulaire logique vise à remettre à jour notre définition précédente de la notion de sous-catégorisation, pour tenir compte des propriétés lexicales des verbes acceptant des sujets ou objets

nominaux ou sous la forme de propositions complétives :

$$\begin{aligned} \textit{sous-catégorisation}(v,s,o) \triangleq\ & \text{avec}(o) \Rightarrow \mathsf{transitif}(v) \\ & \wedge\ \text{sans}(o) \Rightarrow \mathsf{intransitif}(v) \\ & \wedge\ \textit{proposition}(o) \Rightarrow \mathsf{complétive\ objet}(v) \\ & \wedge\ \textit{synt\_nominal}(o) \Rightarrow \mathsf{objet\ nominal}(v) \\ & \wedge\ \textit{proposition}(s) \Rightarrow \mathsf{complétive\ sujet}(v) \\ & \wedge\ \textit{synt\_nominal}(s) \Rightarrow \mathsf{sujet\ nominal}(v) \end{aligned}$$

Cette définition vient replacer celle proposée précédemment pour la relation *sous-catégorisation*.

Contraintes grammaticales Nous disposons maintenant de tous les outils nécessaires pour actualiser et compléter les contraintes logiques portant sur la grammaire support.

La première production conserve sa contrainte $\textit{sous-catégorisation}(v,s,o)$, mais profite de la version actualisée de cette relation, restreignant l'emploi de complétives ou de syntagmes nominaux comme sujet et objet en fonction des propriétés lexicales du verbe.

La quatrième production ($A \longrightarrow \mathsf{pronom} : p$) requiert maintenant un traitement des cas (pronoms sujets et objets), fourni par la contrainte suivante (la variable liée p' désignant la proposition dont le pronom p est un argument) :

$$\begin{aligned} & \mathsf{nominatif}(p) \Rightarrow \exists p'.\textit{sujet}(p',p) \\ \wedge\quad & \mathsf{accusatif}(p) \Rightarrow \exists p'.\textit{objet}(p',p) \end{aligned}$$

Cette double condition évite de sanctionner des énoncés employant des pronoms relatifs dans des rôles syntaxiques incorrects, comme dans « *le mécanisme qui j'ai réparé » (pronom sujet employé comme objet) ou « *la machine que tombe en panne régulièrement » (le cas inverse).

De plus, une structure abstraite valide doit requérir une correspondance unique entre propositions et pronoms relatifs : chaque proposition doit être associée à un unique pronom, et réciproquement. Nous contraignons ainsi un pronom relatif à être lié à une unique relative [2], par le biais d'un chemin *wh* valide :

$$\mathsf{relatif}(p) \Rightarrow \exists!r.\textit{relative}(r) \wedge \textit{chemin\_wh}(r,p)$$

Cette contrainte exclut l'emploi d'un pronom relatif hors du contexte d'une relative, éliminant des phrases telles que « *Que j'ai rencontré. » en forçant l'existence d'une relative r. L'unicité de r est techniquement redondante, au vu des définitions de *relative* et *chemin_wh* : tout nœud constituant un candidat acceptable pour r doit être dominé par une arête *mod*, excluant la possibilité

2. La condition d'unicité dans une quantification existentielle, dénotée par $\exists!$, est définissable en logique du premier ordre comme suit : $\exists!x.P(x) \overset{def}{\Leftrightarrow} \exists x.P(x) \wedge \forall x'.P(x') \Rightarrow x' = x$.

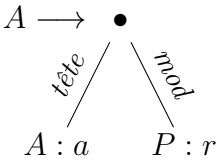
d'être dominé par *objet* et donc celle qu'il existe un autre candidat valable pour r. Cette condition d'unicité fait néanmoins sens dans le cadre de cette contrainte : chaque pronom doit pouvoir être associé à une unique proposition relative.

Enfin, nous imposons l'accord en genre et en nombre d'un pronom avec son antécédent. Remarquons que la définition actuelle d'un antécédent limite pour l'instant l'application de cette règle d'accord aux pronoms relatifs, mais que cette contrainte sur les pronoms resterait valide dans la perspective d'une définition plus large de la notion d'antécédent.

$$\forall a.\mathit{antécédent}(a,p) \Rightarrow \mathit{accord\_genre}(a,p) \wedge \mathit{accord\_nombre}(a,p)$$

Cette règle d'accord élimine des énoncés où le pronom relatif et son antécédent sont incompatibles, tels que « *cette voiture, lesquels revient de chez le garagiste, ... ».

Pour finir, la sixième production permet la modification d'un argument par une relative, et porte deux contraintes : d'une part, une telle modification est généralement interdite sur les pronoms ; d'autre part, chaque relative doit contenir un unique pronom relatif au terme d'un chemin *wh* valide. Ces deux contraintes sont formulées comme suit (nous rappelons en passant la production correspondante) :



1. $\neg\,(\mathsf{pronom}(a) \wedge \mathsf{relatif}(p))$
2. $\exists!p.\mathsf{pronom}(p) \wedge \mathsf{relatif}(p) \wedge \mathit{chemin\_wh}(r,p)$

Ces dernières contraintes excluent d'une part la modification d'un pronom relatif (telle que « *l'employé qui que la société va licencier est parti ») ; et d'autre part l'absence de pronom relatif dans une relative (« *les chaises elles sont sur les bureaux ») ou sa multiplicité (« *l'homme qui que regarde dans le miroir »).

Linéarisations Nous décrivons maintenant les deux linéarisations associées à notre grammaire actualisée, permettant l'antéposition des pronoms relatifs et l'ajout de conjonctions de subordination en surface, et incluant un traitement des évènements en sémantique.

Forme de surface La principale difficulté dans la mise au point de règles de linéarisation pour cette version de la grammaire est le traitement de l'antéposition des pronoms relatifs : leur position dans chaîne constituant la réalisation de surface est en effet arbitrairement distante de leur position dans la structure abstraite. Toutefois, ces pronoms peuvent être commodément insérés lors de la linéarisation : au niveau de la racine de la relative à laquelle ils se rattachent, la réalisation du pronom peut être insérée entre celle de sa

relative (à droite) et celle du syntagme nominal auquel sa relative est rattachée (à gauche). Nous utilisons à cet effet, le mécanisme de requêtes logiques introduit dans la section 3.4.3 : la variable **p** présente dans la sixième règle de linéarisation ci-dessous dénote ainsi la réalisation attachée à toute position p satisfaisant la pré-condition $extrait(p) \wedge chemin\_wh(r, p)$. Le pronom extrait lui-même (situé à la position p) n'est pas réalisé localement en raison de la règle de linéarisation associée à la production qui le domine (la première). Celle-ci associe la chaîne vide ε à un argument formé par un pronom extrait, et préserve la réalisation de l'argument dans le cas contraire. Deux variables (su et ob) sont introduites à cet effet : elles peuvent être envisagées comme une abstraction de la notion de réalisation de l'argument (sujet ou un objet). Observons que ces règles de linéarisation entraînent que la réalisation d'une phrase incorpore exactement une fois chaque pronom, en raison des contraintes de grammaticalité : chaque relative (associée à une requête logique) correspond à exactement un pronom extrait (réalisé localement par ε), et réciproquement.

Les règles de linéarisation ajoutées ou modifiées sont les suivantes :

1. $P \longrightarrow \bullet : p$; tête : verbe : v ; sujet : $A : s$; objet : $(A) : o$; $su\ v\ ob$ où $su = \left\{ \begin{array}{ccc} extrait(s) & \longrightarrow & \varepsilon \\ sinon & \longrightarrow & s \end{array} \right\}$ et $ob = \left\{ \begin{array}{ccc} extrait(o) & \longrightarrow & \varepsilon \\ sinon & \longrightarrow & o \end{array} \right\}$

5. $A \longrightarrow P : c$; **que** c

6. $A \longrightarrow \bullet$; tête : $A : a$; mod : $P : r$; $extrait(p) \wedge chemin\_wh(r, p) \longrightarrow a\ \mathbf{p}\ r$

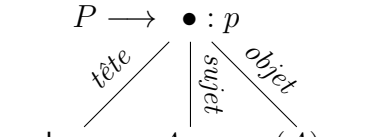
La cinquième règle de linéarisation ajoute explicitement la conjonction de subordination « que » pour introduire une complétive ; cette règle simple suffit à permettre l'analyse de complétives introduites par « que ». Un traitement plus en profondeur des complétives pourrait inclure une lexicalisation des conjonctions de subordination (permettant un traitement approprié en sémantique), et une complétion de la structure abstraite par un nœud supplémentaire, dominé par une arête portant une étiquette *conjonction*.

Linéarisation alternative Les requêtes logiques nous fournissent un moyen de décrire simplement le mouvement du pronom relatif. Elles ne sont cependant pas requises pour rendre compte de ce type de phénomènes dans notre formalisme : l'emploi de lambda-termes plus complexes et de types variés (par contraste avec les réalisations actuelles, qui sont uniformément des

chaînes de type $* \to *$) permet de construire séparément la réalisation locale et l'élément extrait d'une proposition, et de reconstituer l'énoncé par la suite. Spécifiquement, nous pouvons ici construire la réalisation d'une proposition contenant un pronom extrait comme une paire de chaînes, contenant à gauche la réalisation locale de la proposition (où le pronom est absent), et à droite le pronom extrait. La règle de linéarisation combinant la relative et son syntagme nominal peut alors concaténer ces chaînes dans l'ordre approprié pour obtenir l'antéposition du pronom relatif.

En suivant les conventions de notation utilisées jusque là, nous notons (s_1, s_2) dans la règle de linéarisation ci-dessous la paire formée par les chaînes s_1 et s_2 ; et dénotons par $\pi_1(p)$ et $\pi_2(p)$ les projections gauche et droite (respectivement) de la paire p. Ces opérations de construction et projection pouvant être représentées par le lambda-calcul simplement typé (comme illustré dans la section 2.2.4), ces notations constituent simplement une abréviation commode pour les lambda-termes correspondants, et ne requièrent pas d'augmenter le pouvoir d'expression des règles de linéarisation données jusque là.

Les règles de linéarisation suivantes produisent alors les mêmes réalisations que les précédentes, sans employer de requêtes logiques :

1. 

$$\left\{\begin{array}{cc} prop\_ext(p) & \longrightarrow (su\ v\ ob, pr) \\ sinon & \longrightarrow s\ v\ o \end{array}\right\}$$

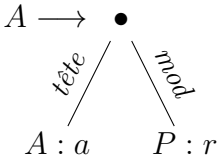
$$où\ \ su = \left\{\begin{array}{cc} extrait(s) & \longrightarrow \varepsilon \\ sinon & \longrightarrow s \end{array}\right\}$$

$$et\ \ ob = \left\{\begin{array}{cc} prop\_ext(o) & \longrightarrow \pi_1(o) \\ extrait(o) & \longrightarrow \varepsilon \\ sinon & \longrightarrow o \end{array}\right\}$$

$$et\ \ pr = \left\{\begin{array}{cc} prop\_ext(o) & \longrightarrow \pi_2(o) \\ extrait(o) & \longrightarrow o \\ extrait(s) & \longrightarrow s \end{array}\right\}$$

5. $A \longrightarrow P : c$

$$\left\{\begin{array}{cc} prop\_ext(c) & \longrightarrow (\mathbf{que}\ \pi_1(c), \pi_2(c)) \\ sinon & \longrightarrow \mathbf{que}\ c \end{array}\right\}$$

6. 

$$a\ \pi_2(r)\ \pi_1(r)$$

Le fonctionnement de ces règles nous paraît cependant moins intuitif que celui des précédentes, dans lesquelles le mécanisme de requête logique permet d'éviter les problèmes de typage, dus à la possibilité pour la réalisation d'un nœud d'être, au choix, une chaîne (de type $* \to *$) ou une paire de chaînes (ayant le type $(\sigma \to \sigma \to \sigma) \to \sigma$, avec $\sigma = * \to *$). Une solution alternative résolvant cette ambigüité pourrait être d'associer à tous les nœuds de la structure abstraite une paire de chaînes, incluant à toute réalisation un

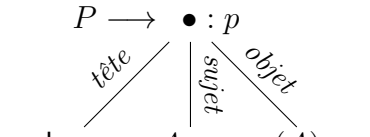
composant extrait (éventuellement vide). Ce choix de modélisation, évoquant l'hypothèse qu'une phrase simple (sans mouvement) contient un composant déplacé « vide », nous paraît cependant difficile à justifier linguistiquement.

Sémantique Notre linéarisation en sémantique pour le traitement des subordonnées complétives passe par l'emploi d'une sémantique évènementielle, à la manière de Davidson [1967]. L'argument d'un verbe sous-catégorisant pour une complétive est un évènement, dénoté par une variable e de type ^{e}, ce même évènement étant également un argument supplémentaire du verbe de la complétive. Par exemple, la sémantique associée à une phrase telle que « Luc croit que le chat reviendra. » est qu'il existe un évènement e que constitue le retour du chat, et que c'est en cet évènement que Luc croit. Accessoirement, l'analyse de cette phrase affirme qu'il existe également un évènement de croyance (dont Luc est l'acteur), dont la continuation n'est pas utilisée. La formule logique associée est $\exists x.\textbf{\textit{chat}}(x) \wedge \exists e_2.\exists e_1.\textbf{\textit{croit}}(e_1, \textbf{\textit{Luc}}, e_2) \wedge \textbf{\textit{reviendra}}(e_2, x)$, où le premier argument d'un prédicat verbal est l'évènement décrit par ce verbe, et les suivants correspondent respectivement à son sujet et à son objet. Ce traitement des complétives n'est pas entièrement satisfaisant par certains aspects (en particulier, il ne permet pas en l'état un traitement approprié des problèmes liés à l'intension), mais constitue néanmoins une base suffisante pour démontrer le fonctionnement de notre formalisme, et permettre dans la suite un traitement adéquat des verbes à contrôle.

Les nouvelles règles de linéarisation sémantique associées aux productions de la grammaire sont les suivantes :

$$\left\{ \begin{array}{ccc} prop\_ext(p) & \longrightarrow & ph \\ sinon & \longrightarrow & ph\ \mathbf{\Omega}^e \end{array} \right\}$$

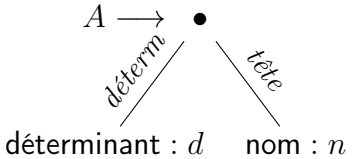
$$où\ \ ph = \lambda r.\lambda C.su\ \lambda x.ob\ \lambda y.\exists e.(C\ e) \wedge nv$$

1. 

$$où\ \ su = \left\{ \begin{array}{ccc} extrait(s) & \longrightarrow & s\ r \\ sinon & \longrightarrow & s \end{array} \right\}$$

$$et\ \ ob = \left\{ \begin{array}{ccc} prop\_ext(o) & \longrightarrow & o\ r \\ extrait(o) & \longrightarrow & o\ r \\ sinon & \longrightarrow & o \end{array} \right\}$$

$$et\ \ nv = \left\{ \begin{array}{ccc} \mathsf{transitif}(v) & \longrightarrow & v\ e\ x\ y \\ sinon & \longrightarrow & v\ e\ x \end{array} \right\}$$

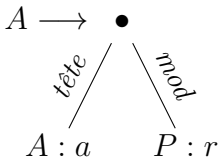
2. 

$$\lambda P.d\ \lambda x.n\ x \wedge P\ x$$

3. $A \longrightarrow \mathsf{nom\ propre} : np$ $\qquad \lambda P.P\ np$

4. $A \longrightarrow \mathsf{pronom} : p$ $\quad \left\{ \begin{array}{ccc} extrait(p) & \longrightarrow & \lambda r.\lambda P.P\ r \\ sinon & \longrightarrow & p \end{array} \right\}$

5. $A \longrightarrow P : c$ $\quad \left\{ \begin{array}{ccc} prop\_ext(c) & \longrightarrow & c \\ sinon & \longrightarrow & c \end{array} \right\}$

6. 

$\lambda P.a\ \lambda x.(r\ x\ \lambda e.\mathsf{T}) \wedge P\ x$

Observons tout d'abord la quatrième règle de linéarisation, associée aux pronoms. Celle-ci propose une linéarisation alternative en cas d'extraction (s'il s'agit d'un pronom relatif) : dans ce denier cas, le type sémantique de la réalisation du pronom est $e \to (e \to t) \to t$, plutôt que $(e \to t) \to t$. Cet élément supplémentaire est celui auquel la propriété P est appliquée, omettant la quantification. En effet, une proposition relative n'est autre qu'un modificateur (de type $e \to t$), s'appliquant à l'élément quantifié par le groupe nominal auquel elle se rattache. Cet élément, qui est manquant dans le contexte de la relative, correspond à la variable r de type e qui est abstraite ici.

Ceci observé, la règle de linéarisation associée à la première production est la plus complexe. Afin d'en simplifier la compréhension, nous avons fait usage du langage de macros (décrit section 3.5.1) pour en isoler les constituants variables (au travers des variables *su*, *ob* et *nv*), et factoriser la majeure partie du terme entre deux réalisations alternatives (grâce à la variable *ph*). Les composants sujet et objet de la phrase, de type $(e \to t) \to t$, sont abstraits au moyen des variables *su* et *ob* respectivement : dans le cas général, ces variables dénotent exactement la réalisation associée au sujet ou à l'objet dans la structure abstraite. Toutefois, dans le cas où l'un des deux est (ou contient) un pronom relatif extrait, l'élément manquant à celui-ci est fourni sous la forme d'une variable libre r, qui dénote l'élément auquel se réfère le pronom relatif. Observons que les contraintes d'îlot formulées plus haut interdisent à une complétive sujet de contenir une extraction dans une phrase valide : la réalisation alternative $prop\_ext(s) \longrightarrow s\ r$ de *su* est donc inutile, et a été exclue de cette règle de linéarisation. Enfin, le noyau verbal, représenté par la variable *nv*, est formé par l'application du prédicat verbal aux variables libres dénotant ses arguments : la variable e représente l'évènement décrit par le verbe, x est son sujet et, si la sous-catégorisation du verbe le réclame, y est son objet.

Ces composants sont ensuite combinés pour former la variable *ph* dénotant le sens de la proposition : une proposition affirme l'existence de l'évènement e apparaissant dans *nv*, et permet une continuation sur celui-ci au moyen de la variable C. Le sujet et l'objet sont ensuite ajoutés au moyen de la construction usuelle par montée de type, et la variable C est ensuite abstraite. Pour terminer la variable r susceptible d'apparaître dans *su* ou *ob* (dans le contexte d'une relative) est à son tour abstraite.

Une fois la variable *ph* formée, deux cas sont envisageables : soit la proposition contient une extraction, auquel cas son type doit être $e \to (e \to t) \to t$ (le premier argument correspond à l'élément dénoté par le pronom relatif, et le second à la continuation de l'évènement), soit ce n'est pas le cas, auquel cas son seul argument possible est la continuation de l'évènement : nous fournissons alors un élément sémantique vide ($\mathbf{\Omega}^e$) en guise de variable r.

Observons finalement que dans ce dernier cas, la variable libre r n'apparaît ni dans *su*, ni dans *ob* ; la constante $\mathbf{\Omega}$ n'apparaît donc jamais dans la réalisation d'une phrase valide. Cette propriété, tout comme le bon typage des expressions *su* et *ob* n'est pas garantie par le formalisme : elle est une conséquence (prouvable) des contraintes de validité associées à la structure abstraite, et des règles de linéarisation fournies. La mise au point d'un système d'annotation de types plus poussé, incluant des conditions logiques (tel qu'évoqué lors de la description du système de typage de notre langage de macro, section 3.5.3), permettrait potentiellement de vérifier ce type de propriétés automatiquement.

Les règles de linéarisation associées à la seconde et à la troisième production demeurent inchangées. La cinquième production, permettant de réécrire un argument comme une complétive, distingue le cas où celle-ci contient une extraction (le type de la variable c est alors $e \to (e \to t) \to t$) de celui où ce n'est pas le cas (auquel cas le type de c est $(e \to t) \to t$). Dans les deux cas, la réalisation associée à l'argument est celle de la complétive qui le compose (c), sans modification. Enfin, la sixième règle de linéarisation modifie un argument a au moyen d'une relative r : ce traitement s'effectue en appliquant a (qui a le type d'un quantificateur) au prédicat appliquant l'élément x (qui sera identifié avec l'élément quantifié par a) à la relative r. Cet élément est ainsi identifié avec la variable r qui apparaissait dans la première règle de linéarisation dans le contexte d'une relative. La réalisation de la relative est ensuite complétée par une continuation vide (substituant la valeur de vérité constante T à la continuation de l'évènement $C\ e$). Enfin, une nouvelle continuation $P\ x$ est ajoutée à la réalisation du syntagme nominal modifié, afin de préserver son type, selon la construction usuelle. Observons que cette règle suppose que la relative est rattachée à un groupe nominal formé (incluant un déterminant), ce qui n'est pas toujours justifié linguistiquement.

4.5 Traitement du contrôle et linéarisations synchrones

Nous souhaitons maintenant inclure à notre traitement des subordonnées des propositions infinitives, introduites par des verbes à contrôle (qualifiés de verbes caténatifs en anglais) ; ces constructions incluent des énoncés tels que « Thomas souhaite partir. ». Sur le plan syntaxique, la proposition infinitive

qui sert d'argument objet à un tel verbe est dénuée de sujet explicite. Du point de vue sémantique, l'argument correspondant à son sujet est l'un des arguments du verbe contrôleur, qui est dupliqué au profit de la subordonnée. Ainsi, dans l'exemple précédent, Thomas est à la fois l'acteur du souhait et celui du départ, bien qu'il n'apparaisse qu'une fois dans la forme de surface.

L'argument dupliqué par le verbe contrôleur n'est pas nécessairement son sujet ; il peut également s'agir d'un autre objet pour lequel le verbe sous-catégorise, comme dans « Thomas demande à Paul de partir. », où Paul est à la fois le destinataire de la demande et l'acteur du départ. L'interprétation sémantique que nous souhaitons donner à ce phénomène est que la proposition contrôlée (la subordonnée infinitive) attend un argument (correspondant à son sujet) et qu'une propriété lexicale du verbe contrôleur (liée à sa sous-catégorisation), « décide » lequel de ses arguments (sujet ou second objet) est dupliqué au bénéfice de sa subordonnée.

Outre cette analyse en sémantique, nous désirons proposer de multiples linéarisations en surface, vers plusieurs langues européennes. En effet, des problèmes uniques liés à l'ordre des mots apparaissent dans des constructions impliquant plusieurs subordonnées infinitives imbriquées par des verbes à contrôle. Ainsi, dans des exemples tels que « Jean ordonne à Marie de demander à Paul de transmettre à Luc de … », chaque verbe (à l'exception du dernier dans l'ordre linéaire) est un verbe à contrôle, doté de deux objets, dont l'un est une subordonnée infinitive, et l'autre un syntagme nominal (plus une éventuelle préposition). Des langues comme le français et l'anglais privilégient dans ces cas un ordre séquentiel, alternant objets nominaux et subordonnées ($s_1 v_1 o_1 v_2 o_2 v_3 o_3 \dots$). L'allemand, en revanche, rejette le verbe d'une proposition subordonnée en dernière position, produisant une construction parenthésée (adoptant de manière canonique la forme $s_1 v_1 o_1 o_2 o_3 \dots v_3 v_2$) ; un certain degré de liberté est toutefois permis dans l'ordre des arguments des subordonnées, requérant un traitement spécifique tel que proposé dans le chapitre 5. Enfin, le néerlandais présente dans ces phrases un phénomène de dépendances croisées en série ($s_1 v_1 o_1 o_2 o_3 \dots v_2 v_3 \dots$). Ce dernier ordre échappe à l'analyse par une grammaire hors-contexte, comme argumenté par Shieber [1985] : bien que la structure abstraite de ce type de construction puisse être exprimée par un langage régulier d'arbres, une linéarisation naïve (qui se contente de concaténer les réalisations des différents sous-nœuds dans un certain ordre) ne peut pas produire la réalisation attendue. Nous emploierons pour le néerlandais des paires de chaînes, décrites et manipulées par le lambda-calcul, afin de surmonter cette difficulté.

Structure abstraite du phénomène de contrôle Afin de modéliser la structure des verbes à contrôle objet, qui sous-catégorisent pour deux compléments distincts, nous introduisons une nouvelle étiquette d'arête obj_2 dans nos structures abstraites. Celles-ci seront partagées entre les trois langues cibles,

qui ne se distingueront que lors du processus de linéarisation. Ce choix de modélisation est quelque peu arbitraire, dans la mesure où la sous-catégorisation des verbes à contrôle peut varier subtilement d'une langue à l'autre : il pourrait être souhaitable de distinguer, par exemple, un objet second accusatif d'un objet datif, indirect [*cf.* Butt *et al.*, 1999, p. 48-51]. Nous nous satisfaisons ici de cette approximation dans la mesure où notre but est d'illustrer l'emploi du formalisme décrit dans le chapitre 3, sans avoir l'ambition de construire une grammaire multilingue à large couverture.

La figure 4.5 détaille la structure abstraite d'un syntagme nominal modifié par une relative contenant des verbes à contrôle imbriqués. La réalisation en français de cet énoncé partiel (qui devrait, selon notre grammaire, être employé comme argument dans une structure plus large) serait la chaîne « un livre que Jean laisse Marie aider Anne à lire ». Bien que contestable en français en raison de l'extraction à travers un complément datif, cette phrase se linéarise correctement dans les langues cibles de la grammaire. Observons que le sujet d'une proposition contrôlée est absent de la structure profonde : il doit en effet être déduit en sémantique de la nature du verbe contrôleur et de ses arguments (c'est Marie qui apporte l'aide, et Anne qui effectue la lecture). Observons également que la linéarisation en sémantique de la relative à droite doit correctement abstraire l'objet du verbe **lire**, et le faire correspondre à l'antécédent (**un livre**) ; et que, comme précédemment, le pronom relatif **que** correspondant doit être antéposé en surface.

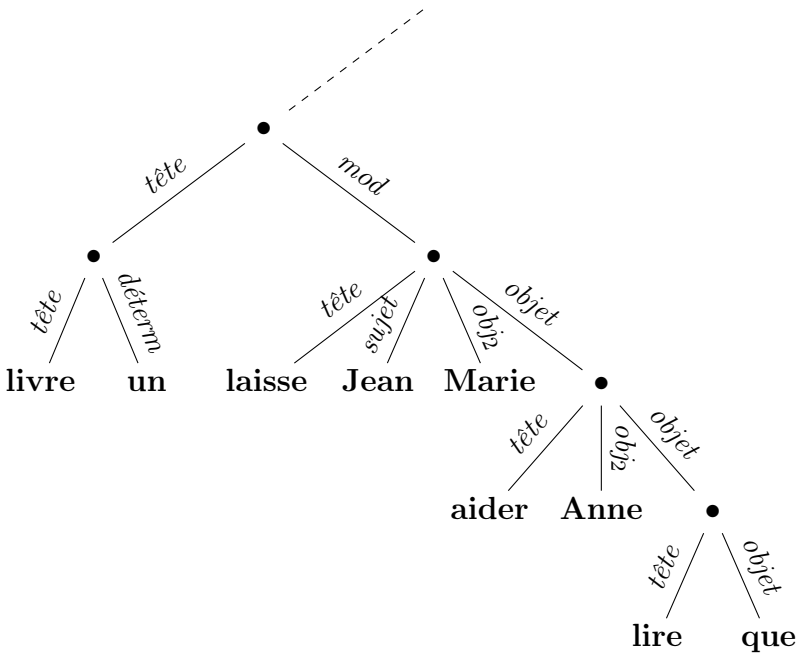


FIGURE 4.5 – Structure d'un syntagme incluant des proposition contrôlées

Mise à jour du lexique Afin de contraindre les structures abstraites de manière appropriée, nous intégrons à notre lexique des verbes à contrôle (sujet et objet) ainsi que la forme infinitive des verbes existants. Les propriétés associées à ces entrées sont respectivement contrôle sujet, contrôle objet et infinitif. Notons que, comme pour les noms dont les propriétés d'accord sont sous-spécifiées, un même verbe peut apparaître plusieurs fois dans le lexique, avec ou sans les propriétés contrôle sujet et contrôle objet. Tout comme la structure abstraite des énoncés, les entrées lexicales seront communes aux trois langues cibles. Chaque entrée disposera cependant, outre de sa réalisation en sémantique, de trois réalisations distinctes en surface, correspondant à chacune des linéarisations synchrones.

La table 4.6 propose un ensemble d'entrées lexicales liées au contrôle, ainsi que les réalisations qui leur sont attachées.

Entrée	Propriétés	en	de	nl
vouloir	verbe, infinitif, transitif, contrôle sujet	want	willen	wilen
aider	verbe, infinitif, transitif, contrôle objet	help	helpen	helfen
laisse	verbe, transitif, contrôle objet	lets	lässt	laat
lire	verbe, infinitif, transitif, objet nominal, complétive objet	read	lesen	lezen

TABLE 4.6 – Entrées lexicales liées au contrôles et réalisations

Vocabulaire logique Nous introduisons deux nouveaux prédicats permettant de spécifier des contraintes de grammaticalité sur les verbes à contrôle. Le premier caractérise l'ensemble des verbes à contrôle en regroupant les propriétés lexicales contrôle sujet et contrôle objet ; le second dénote une proposition contrôlée (tenant lieu d'argument objet pour un verbe contrôleur).

$$\textit{contrôleur}(x) \triangleq \text{contrôle sujet}(x) \vee \text{contrôle objet}(x)$$
$$\textit{contrôlée}(x) \triangleq \exists v.\textit{contrôleur}(v) \wedge \textit{tête} \uparrow \textit{objet}(v, x)$$

De plus, les linéarisations vers l'allemand et le néerlandais requièrent un ordre des mots différent entre propositions indépendantes et subordonnées (le verbe étant, pour ces dernières, rejeté à la fin). Nous obtiendrons les réalisations attendues en incluant des conditions appropriées dans les règles de linéarisation, au moyen des prédicats *indépendante* et *subordonnée*, définis comme suit : dans notre grammaire, une proposition est indépendante si elle n'est dominée par aucune autre proposition, et subordonnée dans le cas contraire.

$$\textit{indépendante}(x) \triangleq \textit{proposition}(x) \wedge \nexists p.\textit{proposition}(p) \wedge \text{dom}(p, x)$$
$$\textit{subordonnée}(x) \triangleq \textit{proposition}(x) \wedge \neg \textit{indépendante}(x)$$

Grammaire support et contraintes logiques Nous sommes maintenant en mesure d'actualiser les productions de notre grammaire, ainsi que les contraintes logiques associées, pour inclure les phénomènes de contrôle.

La principale modification porte sur la première production, permettant l'ajout d'un argument objet supplémentaire (optionnel, étiqueté $o2$) et rendant l'argument sujet optionnel (celui-ci étant absent dans le cas d'une infinitive contrôlée). La nouvelle version de la première production est illustrée par la figure 4.6.

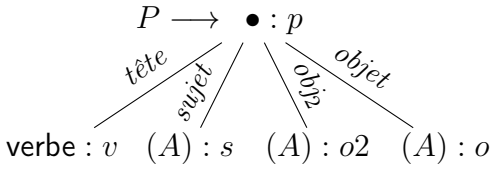


FIGURE 4.6 – Première production actualisée pour le contrôle

Par suite, nous modifions les contraintes logiques portant sur cette production, en incluant tout d'abord deux contraintes supplémentaires portant sur les propositions infinitives. D'une part, dans le cadre de notre grammaire, les propositions infinitives coïncident avec les propositions subordonnées contrôlées :

$$\textit{contrôlée}(p) \Leftrightarrow \mathsf{infinitif}(v)$$

D'autre part, en l'absence de sujet, la contrainte d'accord sujet-verbe est trivialement satisfaite. Nous conditionnons donc son application à l'existence d'un sujet de la manière suivante :

$$\text{avec}(s) \Rightarrow \textit{accord\_nombre}(s, v) \wedge \textit{accord\_personne}(s, v)$$

Enfin, nous modifions le prédicat *sous-catégorisation* pour traiter le cas des verbes contrôleurs, en tenant compte du second objet, de l'optionnalité du sujet et des complétives infinitives :

— Un sujet est absent si et seulement si son verbe est à l'infinitif.
— Un objet second est l'argument d'un verbe à contrôle objet, et doit être nominal.
— Un verbe peut sous-catégoriser pour une complétive objet parce qu'il est un verbe à contrôle, ou parce qu'il possède la propriété lexicale spécifique complétive objet.

La définition actualisée du prédicat résultant (d'arité 4) est donnée ci-dessous, chaque ligne correspondant à une « règle » de sous-catégorisation. Les trois conditions spécifiées ci-dessus sont implémentées respectivement par la première, la troisième et la quatrième ligne, le reste de la définition demeurant inchangé. En outre, la contrainte de bonne formation imposée précédemment selon laquelle les arguments de la première production doivent satisfaire le prédicat *sous-catégorisation* est maintenue.

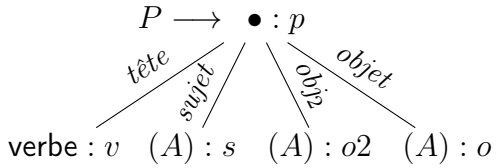
$$\begin{aligned}\textit{sous-catégorisation}(v, s, o, o2) \triangleq\ & \text{sans}(s) \Leftrightarrow \textsf{infinitif}(v)\\ & \wedge\ \text{avec}(o) \Rightarrow \textsf{transitif}(v) \wedge \text{sans}(o) \Rightarrow \textsf{intransitif}(v)\\ & \wedge\ \text{avec}(o2) \Leftrightarrow \textsf{contrôle objet}(v) \Leftrightarrow \textit{synt\_nominal}(o2)\\ & \wedge\ \textit{proposition}(o) \Rightarrow \textsf{complétive objet}(v) \vee \textit{contrôleur}(v)\\ & \wedge\ \textit{synt\_nominal}(o) \Rightarrow \textsf{objet nominal}(v)\\ & \wedge\ \textit{proposition}(s) \Rightarrow \textsf{complétive sujet}(v)\\ & \wedge\ \textit{synt\_nominal}(s) \Rightarrow \textsf{sujet nominal}(v)\end{aligned}$$

Linéarisation en sémantique Pour la plupart des productions, les règles de linéarisation sémantique restent inchangées, hormis l'avant dernière, qui doit tenir compte de contraintes de typage supplémentaire : un argument formé par une complétive peut en effet avoir le type $e \to e \to (e \to t) \to t$: le second e manquant à gauche représente alors, comme précédemment, l'élément associé à un pronom relatif ; et le premier correspond au sujet d'une proposition infinitive contrôlée.

Ce mécanisme s'explique en examinant la règle de linéarisation associée à la première production, donnée figure 4.7. Celle-ci est passablement complexe, mais le langage de macros adopté dans le chapitre 3 permet de gérer cette complexité, en cloisonnant les différents composants de la linéarisation afin de les considérer indépendamment.

Remarquons tout d'abord que cette règle de linéarisation conserve le même schéma que précédemment, son terme principal étant factorisé par la variable *ph*. Ce terme abstrait la continuation C de l'évènement dénoté par la proposition, puis emploie la construction habituelle par montée de type pour traiter la quantification de chacun de ses trois arguments, *su*, *ob* et $o2$. Observons que l'objet second $o2$ est traité avant l'objet *ob*. Enfin, le terme *ph* quantifie sur l'évènement e associée à la proposition, applique sa continuation, puis incorpore la sémantique du noyau verbal *nv*. Ce dernier applique simplement le terme représentant le sens du verbe à ses arguments, en fonction de ses contraintes de sous-catégorisation. Remarquons que, sémantiquement, tout verbe possède au moins un argument en plus de son évènement, même s'il s'agit d'un infinitif dénué de sujet en surface.

Examinons maintenant chacun des arguments (*su*, *ob* et $o2$) séparément. Par défaut, l'argument sujet *su* est simplement la réalisation s associée au sous-terme correspondant. Comme précédemment, il peut également s'agir d'un pronom (relatif) extrait, auquel cas cas la réalisation emploie la variable libre r qui conserve le même usage que précédemment. Enfin, il est possible que la proposition soit contrôlée, auquel cas l'argument sujet du verbe est déterminé (et quantifié) par la proposition parente ; dans ce dernier cas, la variable libre s' est employée, d'une manière similaire à la variable r pour l'élément désigné par un pronom relatif.



$$\left\{ \begin{array}{c} contrôlée(p) \longrightarrow \left\{ \begin{array}{cc} prop\_ext(p) & \longrightarrow \lambda s'.\lambda r.ph \\ sinon & \longrightarrow \lambda s'.(\lambda r.ph)\ \mathbf{\Omega}^e \end{array} \right\} \\ \\ sinon \longrightarrow \left\{ \begin{array}{cc} prop\_ext(p) & \longrightarrow \lambda r.(\lambda s'.ph)\ \mathbf{\Omega}^e \\ sinon & \longrightarrow (\lambda r.\lambda s'.ph)\ \mathbf{\Omega}^e\ \mathbf{\Omega}^e \end{array} \right\} \end{array} \right\}$$

$où\ \ ph = \lambda C.su\ \lambda x.o2\ \lambda z.ob\ \lambda y.\exists e.(C\ e) \wedge nv$

$$où\ \ su = \left\{ \begin{array}{cc} contrôlée(p) & \longrightarrow \lambda P.P\ s' \\ extrait(s) & \longrightarrow s\ r \\ sinon & \longrightarrow s \end{array} \right\}$$

$$et\ \ ob\ = \left\{ \begin{array}{c} extrait(o) \longrightarrow o\ r \\ \\ prop\_ext(o) \longrightarrow \left\{ \begin{array}{cc} \textsf{contrôle objet}(v) & \longrightarrow o\ x\ r \\ \textsf{contrôle sujet}(v) & \longrightarrow o\ z\ r \\ sinon & \longrightarrow o\ r \end{array} \right\} \\ \\ sinon \longrightarrow \left\{ \begin{array}{cc} \textsf{contrôle objet}(v) & \longrightarrow o\ x \\ \textsf{contrôle sujet}(v) & \longrightarrow o\ z \\ sinon & \longrightarrow o \end{array} \right\} \end{array} \right\}$$

$$et\ \ nv\ = \left\{ \begin{array}{cc} \textsf{contrôle objet}(v) & \longrightarrow v\ e\ x\ y\ z \\ sinon & \longrightarrow \left\{ \begin{array}{cc} \textsf{transitif}(v) & \longrightarrow v\ e\ x\ y \\ sinon & \longrightarrow v\ e\ x \end{array} \right\} \end{array} \right\}$$

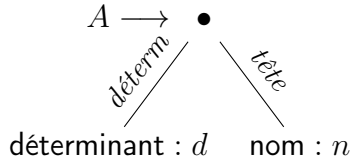
FIGURE 4.7 – Linéarisation sémantique associée à la première production

Considérons maintenant l'argument objet *ob* : il fait l'objet du même traitement que précédemment pour l'extraction, distinguant les cas *extrait*(o) et *prop_ ext*(o) où l'objet requiert un argument supplémentaire r se rapportant au pronom relatif. Cependant, un argument supplémentaire est inséré dans le cas où v est un verbe contrôleur : celui-ci doit en effet dupliquer son sujet (x) ou son objet second (z) au profit de sa complétive infinitive, selon le type de contrôle exercé par le verbe.

Enfin, le dernier argument $o2$ est employé directement : nous ne traitons ici pas la possibilité pour ce dernier de faire l'objet d'une extraction. Le traitement de ce cas par une extension de la grammaire actuelle ne présente pas de difficulté particulière, hormis sur un point délicat : la correspondance supposée jusqu'à présent entre la fonction profonde d'un pronom relatif (*sujet*, *objet*) et sa forme dans le lexique (nominatif, accusatif) constitue une approximation qui atteint ici ses limites. Pour certains verbes, le pronom requerrait en français l'ajout de la préposition « à » suggérant un datif, alors que la forme accusative simple est préférée ailleurs (en particulier pour les verbes de perception comme voir/entendre).

Enfin, considérons le contexte dans lequel le terme *ph* apparaît : celui-ci tient compte des deux éléments susceptibles d'être manquants. Si la proposition p est contrôlée, alors la variable libre s', correspondant à son sujet manquant, doit être abstraite dans la réalisation de la proposition. En outre, comme précédemment, la réalisation doit également abstraire r lorsqu'une extraction est en jeu. La réalisation d'une proposition a donc le type $e^k \to (e \to t) \to t$, où $k \in \{0, 1, 2\}$: si $k = 0$, la proposition est complète ; si $k = 1$, elle fait l'objet soit d'un contrôle, soit d'une extraction, et la variable correspondante est abstraite ; enfin, si $k = 2$, la proposition est contrôlée et fait également l'objet d'une extraction, et les deux variables correspondantes sont laissées abstraites. Observons dans ce cas que l'ordre de ces abstractions (s' puis r) correspond à l'ordre d'application des arguments dans la construction de *ob* (traitant d'abord l'argument lié au contrôle, puis celui des extractions).

Les autres règles de linéarisation demeurent inchangées, à l'exception de la cinquième, qui doit tenir compte du type de son unique variable c. Nous distinguons pour cela quatre cas, liés à la présence ou à l'absence d'une abstraction due au contrôle ou au pronom relatif. Seul le type de la variable c varie d'un cas à l'autre, le terme lui-même demeurant inchangé.

2. 

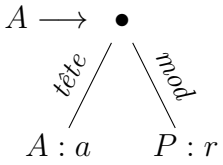
$\lambda P.d\ \lambda x.n\ x \wedge P\ x$

3. $A \longrightarrow$ nom propre : np — $\lambda P.P\ np$

4. $A \longrightarrow$ pronom : p — $\left\{ \begin{array}{ccc} extrait(p) & \longrightarrow & \lambda r.\lambda P.P\ r \\ sinon & \longrightarrow & p \end{array} \right\}$

5. $A \longrightarrow P : c$

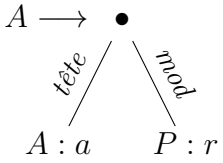
$$\left\{ \begin{array}{ll} \textit{contrôlée}(c) & \longrightarrow \left\{ \begin{array}{ll} \textit{prop\_ ext}(c) & \longrightarrow c \\ \textit{sinon} & \longrightarrow c \end{array} \right\} \\ \textit{sinon} & \longrightarrow \left\{ \begin{array}{ll} \textit{prop\_ ext}(c) & \longrightarrow c \\ \textit{sinon} & \longrightarrow c \end{array} \right\} \end{array} \right\}$$

6. 

$\lambda P.a\ \lambda x.(r\ x\ \lambda e.\top) \wedge P\ x$

Linéarisations synchrones vers la forme de surface Nous décrivons maintenant trois ensembles de règles de linéarisation, afin de produire la forme de surface attendue pour nos structures abstraites modélisant le contrôle en anglais, allemand et néerlandais, en respectant les contraintes d'ordre des mots mentionnées précédemment.

La plupart des règles de linéarisation sont communes aux trois langues et demeurent inchangées : les productions 2, 3 et 4 conservent les règles de linéarisation précédentes, qui concatènent simplement les entrées lexicales dans l'ordre usuel. La sixième règle de linéarisation, rappelée ci-dessous est également inchangée, employant une requête logique pour antéposer le pronom à la relative :

6. 

$\textit{extrait}(p) \wedge \textit{chemin\_ wh}(r, p) \longrightarrow a\ \mathbf{p}\ r$

La cinquième production, qui introduisait manuellement la conjonction de subordination **que**, doit être modifiée : les propositions infinitives (contrôlées) ne requièrent pas de conjonction. En outre, la conjonction employée dépend de la langue cible (**that**, **dass** ou **dat** respectivement) :

5. $A \longrightarrow P : c$

$$\left\{ \begin{array}{ll} \textit{contrôlée}(c) & \longrightarrow c \\ \textit{sinon} & \longrightarrow \textbf{that}/\textbf{dass}/\textbf{dat}\ c \end{array} \right\}$$

Enfin, la première production est celle qui construit les propositions, et doit répondre au problème de l'ordre des mots. Nous détaillons maintenant la règle de linéarisation associée pour chacune des trois langues cibles. Afin d'expliciter les réalisations attendues pour chacun des cas, la figure 4.8 détaille la forme de surface résultant d'une même structure abstraite en anglais, allemand et néerlandais (en reprenant l'exemple de structure abstraite donné par la figure 4.5).

Les mots obtenu lors de la réalisation des feuilles de la structure abstraite sont soulignés, et regroupés selon la proposition dont ils dépendent immédiatement. La flèche en pointillés illustre le mouvement du pronom extrait (correspondant au pronom relatif **que** en français).

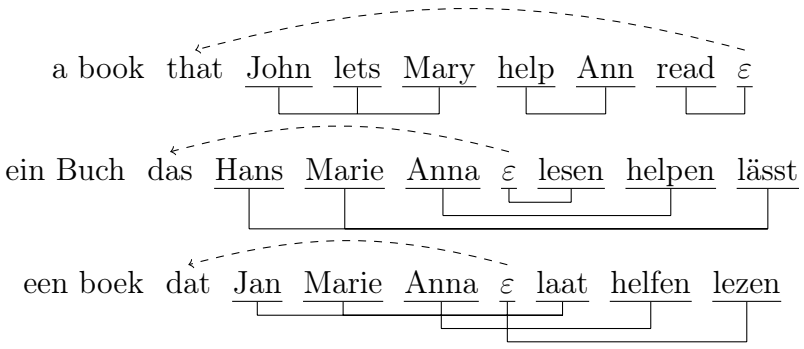


FIGURE 4.8 – Réalisations multilingues et ordre des mots

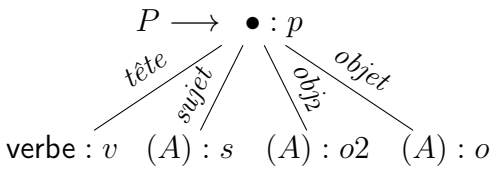
Anglais La règle de linéarisation pour l'anglais est similaire à celle adoptée jusqu'ici. L'ordre canonique place le second complément d'un verbe contrôleur en première position, suivi de la subordonnée infinitive tenant lieu d'objet. La règle de linéarisation résultante est la suivante :

1.
$$P \longrightarrow \bullet : p \qquad su\ v\ o2\ ob$$
$$\text{verbe} : v \quad (A) : s \quad (A) : o2 \quad (A) : o$$
(*tête* : v, *sujet* : $(A) : s$, *obj2* : $(A) : o2$, *objet* : $(A) : o$)
$$où\ su = \left\{ \begin{array}{cc} extrait(s) & \longrightarrow \varepsilon \\ sinon & \longrightarrow s \end{array} \right\}$$
$$et\ ob = \left\{ \begin{array}{cc} extrait(o) & \longrightarrow \varepsilon \\ sinon & \longrightarrow o \end{array} \right\}$$

Allemand Pour notre grammaire, dans le cadre d'une proposition indépendante, l'ordre des mots allemand est identique à celui de l'anglais ; plaçant le verbe en deuxième position, à la suite de son sujet. En revanche, l'allemand impose de rejeter le verbe à la fin dans le contexte d'une subordonnée : ainsi, l'ordre des composants dans une proposition telle que « Er gewann das Rennen. » (« Il a gagné la course. ») devient « Ich weiss, dass er das Rennen gewann. » (« Je sais qu'il a gagné la course. »).

L'analyse du contexte dans lequel une proposition est réalisée (subordonnée ou indépendante) s'effectue au moyen d'une condition de réalisation simple, sélectionnant l'ordre des mots approprié :

1.



$$\left\{ \begin{array}{ccc} \textit{indépendante}(p) & \longrightarrow & \textit{su}\ v\ \textit{o2}\ \textit{ob} \\ \textit{subordonnée}(p) & \longrightarrow & \textit{su}\ \textit{o2}\ \textit{ob}\ v \end{array} \right\}$$

$$\textit{où}\ \ \textit{su} = \left\{ \begin{array}{ccc} \textit{extrait}(s) & \longrightarrow & \varepsilon \\ \textit{sinon} & \longrightarrow & s \end{array} \right\}$$

$$\textit{et}\ \ \textit{ob} = \left\{ \begin{array}{ccc} \textit{extrait}(o) & \longrightarrow & \varepsilon \\ \textit{sinon} & \longrightarrow & o \end{array} \right\}$$

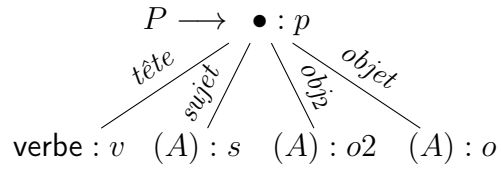
Observons que cette règle ne produit que l'ordre canonique pour les arguments du verbe : des réalisations multiples permettraient d'inclure les autres ordres possibles (autorisant, par exemple, à mettre l'accent sur l'objet du verbe en le plaçant en position initiale). D'autres phénomènes plus complexes, autorisant le mélange entre des arguments appartenant à plusieurs propositions distinctes, seront évoqués et traités dans le prochain chapitre.

Néerlandais L'ordre des mots en néerlandais est en général similaire à celui de l'allemand, à l'exception des constructions impliquant plusieurs verbes à contrôle imbriqués. Dans ce cas, la séquence formée par les verbes rejetés à la fin est inversée : les verbes de chaque subordonnée apparaissent dans l'ordre où leurs propositions ont été introduites (et non dans l'ordre inverse).

Ce phénomène, connu sous le nom de dépendances croisées en série, ne permet pas à une règle de linéarisation simple (choisissant simplement un ordre pour les composants de chaque production) de produire le résultat attendu. Une analyse commode de ce phénomène consiste à produire non pas une chaîne, mais une paire de chaînes en guise de réalisation : le premier élément de la paire contient les arguments des subordonnées, concaténés par ordre d'apparition ; tandis que le second accumule simultanément les verbes, dans le même ordre. La proposition contrôleuse initiale peut alors produire la réalisation attendue en concaténant directement les deux éléments de la paire.

Nous détaillons maintenant la règle de linéarisation que nous associons aux propositions en néerlandais, donnée par la figure 4.9.

Les deux premières alternatives de réalisation, en l'absence de contrôle, sont identiques à celles données pour l'allemand : l'ordre SVO prévalant dans une proposition indépendante, et SOV dans une subordonnée. Par suite, une proposition subordonnée contrôlée a pour réalisation une paire de chaînes, dont le composant gauche est formé par ses arguments objets placés dans l'ordre usuel (avec une éventuelle complétive en dernier), et le composant droit contient le verbe. Une proposition contrôlée étant une infinitive (et donc dénuée



$$
\left\{
\begin{array}{l}
\neg \mathit{contrôleur}(v) \longrightarrow \\
\left\{
\begin{array}{ll}
\mathit{indépendante}(p) \longrightarrow & \mathit{su}\ v\ o2\ \mathit{ob} \\
\mathit{subordonnée}(p) \longrightarrow & \left\{
\begin{array}{ll}
\neg \mathit{contrôlée}(p) \longrightarrow & \mathit{su}\ o2\ \mathit{ob}\ v \\
\mathit{contrôlée}(p) \longrightarrow & (o2\ \mathit{ob}, v)
\end{array}
\right\}
\end{array}
\right\} \\
\mathit{contrôleur}(v) \longrightarrow \\
\left\{
\begin{array}{ll}
\mathit{subordonnée}(p) \longrightarrow & \left\{
\begin{array}{ll}
\neg \mathit{contrôlée}(p) \longrightarrow & \mathit{su}\ o2\ \pi_1(o)\ v\ \pi_2(o) \\
\mathit{contrôlée}(p) \longrightarrow & (o2\ \pi_1(o), v\ \pi_2(o))
\end{array}
\right\} \\
\mathit{indépendante}(p) \longrightarrow & \mathit{su}\ v\ o2\ \pi_1(o)\ \pi_2(o)
\end{array}
\right.
\end{array}
\right.
$$

$$
\text{où } \mathit{su} = \left\{ \begin{array}{lll} \mathit{extrait}(s) & \longrightarrow & \varepsilon \\ \mathit{sinon} & \longrightarrow & s \end{array} \right\} \text{ et } \mathit{ob} = \left\{ \begin{array}{lll} \mathit{extrait}(o) & \longrightarrow & \varepsilon \\ \mathit{sinon} & \longrightarrow & o \end{array} \right\}
$$

FIGURE 4.9 – Règle linéarisation générale des propositions en néerlandais

de sujet), son argument s est nécessairement ε ; nous avons donc omis ce dernier de la linéarisation.

L'alternative de réalisation suivante couvre le cas d'un verbe contrôleur dans une subordonnée : celui ci concatène ses arguments en plaçant d'abord les arguments du verbe (sujet et objet local), puis la séquence d'arguments fournis par sa complétive objet (la première projection de sa réalisation, contenant ses arguments ainsi que ceux de ses éventuelles subordonnées dans l'ordre attendu), puis son verbe, suivi de la séquence de verbes fournis par sa complétive objet (la seconde projection, contenant le verbe de cette dernière et les verbes de ses éventuelles subordonnées).

Le cas suivant couvre les étapes intermédiaires dans une séquence de verbes à contrôle : si une proposition est à la fois contrôleuse et contrôlée, sa réalisation est une paire, dont le premier élément concatène la projection gauche de sa subordonnée à ses arguments, et le second élément concatène la projection droite de sa subordonnée à son verbe. Cette réalisation conserve ainsi les propriétés attendues, à savoir que la réalisation d'une proposition contrôlée est une paire, dont la composante gauche contient les arguments et la composante droite contient les verbes des subordonnées infinitives (contrôlées) successives.

Enfin, la dernière alternative permet de traiter le cas où une proposition indépendante contrôle sa subordonnée : le verbe de la proposition principale est alors placé en seconde position (et non ajouté au début de la séquence de verbe finale), et le reste des arguments est placé dans l'ordre attendu.

4.6 Conclusion

Dans ce chapitre, nous avons mis en pratique le formalisme décrit dans le chapitre précédent, afin de concevoir et de faire évoluer une grammaire couvrant des phénomènes linguistiques divers en syntaxe et en sémantique, parmi lesquels l'accord, le mouvement *wh*, le contrôle et l'ordre des mots. Dans une large mesure, les outils de description proposés se sont avérés adéquats dans le cadre d'un travail de conception grammaticale portant sur des phénomènes non-triviaux. Ils permettent la mise en œuvre des concepts linguistiques abordés, et la concision des descriptions résultantes contribue à la maintenabilité de la grammaire résultante, de même que le nommage des formules logiques (utilisant l'opérateur $\triangleq$) et de certains lambda-termes apparaissant dans les règles de linéarisation (au moyen de notre langage de macros). L'emploi de requêtes logiques facilite en outre la description des phénomènes liés au mouvement.

Une observation intéressante soulevée par l'emploi de ce formalisme est l'existence de nombreuses similarités entre la conception d'une grammaire enrichie et de ses linéarisations, et la programmation en général :

— la relative indépendance entre les différents composants d'une grammaire enrichie (et les éventuelles linéarisations associées) évoque la décomposition d'un programme en modules ;

— le nommage des différents composants d'une grammaire (propriétés lexicales, prédicats et relations ajoutés au vocabulaire logique, étiquettes décorant les productions et variables apparaissant dans les règles de linéarisation) bénéficie de l'intuition fournie par les règles en usage pour le choix des identifiants dans un programme ;

— bien que le formalisme soit conçu pour faciliter l'implémentation et la compréhension de phénomènes linguistiques, les formules logiques et règles de linéarisation les plus complexes bénéficient grandement de l'ajout de commentaires, sous la forme d'annotation expliquant divers choix de mise en œuvre ;

— certains concepts, tels que l'opposition entre une implémentation statique (plus simple) ou dynamique (plus correcte et adaptable) ressurgissent, par exemple dans notre modélisation de l'accord, où toutes les informations morphologiques sont pré-compilées dans le lexique (*cf.* table 4.4), plutôt que calculées dynamiquement d'après un ensemble de règles de linéarisation morphologiques ;

— l'exercice d'une discipline stricte vis-à-vis des types donnés aux réalisations des non-terminaux (par les règles de linéarisation associées aux productions) contribue grandement à garantir la cohérence sémantique des réalisations – à l'inverse, certaines réalisations imprévues (et non souhaitables) peuvent être exclues en vérifiant les types associés à certaines règles de linéarisation (c'est particulièrement le cas pour la linéarisation sémantique du contrôle (figure 4.7), où une proposi-

tion complétive peut avoir quatre types différents en fonction de son contexte) ;
— certaines erreurs susceptibles d'apparaître lors de la modification d'une grammaire (par exemple, lors de la mise à jour de la définition d'un prédicat logique à la lumière de nouvelles informations linguistiques) peuvent être prévenues par l'emploi d'assertions dans les contraintes logiques : en spécifiant plusieurs contraintes partiellement ou totalement redondantes compte tenu des autres contraintes et de la grammaire support, le langage abstrait peut devenir vide de tout énoncé si une modification invalide de manière inattendue une supposition antérieure. Par exemple, notre définition du prédicat *prop_ ext*(x) (p. 89) précise, inutilement, que x doit être une proposition ; mais tout changement dans la définition de la relation *chemin_ wh* qui y inclurait l'arête *mod* risquerait d'entraîner, de façon inattendue, qu'un syntagme nominal soit considéré comme une proposition contenant une extraction, introduisant un « bug » dans la grammaire. D'autres exemples incluent la contrainte d'unicité portant sur la proposition relative associée à chaque pronom relatif (p. 90) ; ou, négativement, notre définition d'une proposition indépendante (p. 99, qui ne doit être dominée par aucun *nœud* dans la structure abstraite, mais devrait en toute rigueur n'être dominée par aucune *proposition*) ; ou encore celle d'un syntagme à la troisième personne (p. 83, qui suppose, peut-être trop arbitrairement, qu'un pronom personnel ne pourra jamais être modifié ou déterminé dans la structure abstraite) ;
— le langage de macros décrit section 3.5.1 dans les règles de linéarisation permet de factoriser certaines constructions, mais soulève les mêmes question de lisibilité que la factorisation de fragments de code : un compromis doit être établi entre la répétition d'un lambda-terme dont une petite partie change d'un copier-coller à l'autre, et une généralisation excessive compliquant inutilement une règle de linéarisation ;
— enfin, le développement d'une grammaire dans le but de sanctionner ou d'exclure certains exemples ou contre-exemples linguistiques, tels que ceux proposés pour expliciter les contraintes d'îlot (p. 87), ressemble de très près à la méthodologie du développement piloté par les tests (*Test-Driven Developement*) en ingénierie logicielle.

Limites et discussion À l'usage, toutefois, certaines limitations dans les capacités du formalisme deviennent apparentes. Certaines redondances dans les règles de linéarisation ne peuvent pas être factorisées par notre langage de macros, en raison de son manque de granularité sur les conditions : les formules logiques apparaissant dans les conditions ne peuvent qu'être fusionnées deux à deux. Par conséquent des termes qui ne diffèrent que par l'emploi d'une variable dans une pré-condition, comme *su* et *ob* dans les modélisations impliquant le

mouvement (p. 92), ne peuvent être factorisés ; il s'agit toutefois d'un problème assez mineur.

Une autre source de lourdeur, due à la discipline de typage choisie pour nos lambda-termes, est la présence de règles de linéarisation telles que celle de la cinquième production en sémantique page 103 : dans tous les cas, la réalisation attachée à la production est la même (c). Cependant, son type peut varier en fonction du contexte, conduisant à employer la variable c avec des types différents sous différentes pré-conditions. Le fait de forcer cette distinction aide à garantir la cohérence de notre réalisation sémantique (en rendant explicites les suppositions faites sur le type des non-terminaux), mais oblige dans ces cas à produire des règles de linéarisation en apparence chargées, alors que leur réalisation est dans tous les cas égale à celle de leur unique argument. L'emploi d'un système de typage plus riche, tel qu'évoqué à la fin du chapitre précédent (voir page 3.5.3), pourrait résoudre ce problème et faciliter l'apport d'une solution au précédent.

Les règles de linéarisation en sémantique pourraient également bénéficier d'une simplification supplémentaire : elles sont actuellement l'élément le plus complexe apparaissant dans nos modélisations (figure 4.7), en grande partie à cause du traitement usuel des quantifications dans la sémantique de Montague. L'emploi de constructions à montée de type, ou plus particulièrement de continuations, répond à des besoins de compositionnalité issus d'autres traditions grammaticales. Il pourrait être opportun de les éliminer là où c'est possible, en mettant à contribution notre mécanisme de requêtes logiques. La réalisation d'un syntagme ne rendrait ainsi compte de sa quantification qu'une fois celui-ci complètement formé. Une relative perte de compositionnalité s'ensuivrait, en ce sens qu'il ne serait plus possible de déterminer complètement le sens d'un fragment partiel de la structure abstraite ; cependant, il peut sembler étrange que le sens d'un syntagme nominal incorpore une continuation se rapportant au sens du verbe à venir.

Pour terminer, le compte rendu qui est fait ici du phénomène de contrôle et de l'ordre des mots résultant est quelque peu superficiel. D'une part, contrairement à ce que nous avons supposé, la propriété de contrôle n'est pas complètement trans-linguistique : certains énoncés n'ont pas d'équivalent direct dans une langue voisine (par exemple, « John wants Mary to stay. » ne peut pas se traduire littéralement en français par une infinitive contrôlée), et nos arguments *objet* et obj_2 ne tiennent pas compte des prépositions qui sont requises par certains verbes pour leurs arguments (infinitif avec ou sans *to* en anglais, objets indirects construits avec *à* en français, *etc.*). Enfin, les exemples illustrant les différences dans l'ordre des mots sont basés sur des constructions complexes, et peuvent par conséquent sembler artificiels, bien qu'ils transcrivent des phénomènes clairement attestés.

Certains de ces derniers problèmes semblent être inhérents au concept de grammaire synchrone multilingue. D'autres pourraient être couverts par notre

grammaire en énumérant et traitant toutes les constructions attestées (élargissement du lexique et de ses information de valence ainsi que de la relation *sous-catégorisation*(), traitement spécifique des verbes de perception, *etc.*), et n'apporteraient pas grand chose de plus au propos de ce chapitre. En revanche, certains phénomènes d'ordre des mots posent des problèmes plus sérieux. C'est en particulier le cas de l'allemand, qui permet le mélange des arguments entre plusieurs propositions complétives imbriquées, un phénomène connu sous le nom de *scrambling*, que notre grammaire actuelle ne couvre pas. L'ensemble des réalisations résultantes, énumérant tous les ordres possibles entre les arguments des subordonnées, ne peut être directement obtenu par les mécanismes de linéarisation utilisés jusqu'ici, et pose un problème de fond.

Plus généralement, d'autres langues offrant un grand degré de liberté dans l'ordre des mots requièrent une énumération exhaustive des ordres possibles, qui est peu commode à exprimer par nos règles de linéarisation. Afin de répondre à ces difficultés, nous introduisons dans le prochain chapitre un mécanisme de représentation compacte, dénotant des phrases modulo permutation de certains mots ou groupes de mots, et étudions ses propriétés algorithmiques. Les termes qui en résultent, représentables par le lambda-calcul, constituent une solution possible aux problèmes d'ordre des mots posés par des phénomènes comme le *scrambling*.

Chapitre 5

Ordres libres

La méthodologie de formalisation linguistique adoptée dans le chapitre précédent permet de produire aisément des linéarisations vers la forme canonique des phrases dans les langues où l'ordre des mots est fixé. Toutefois, elle échoue à représenter de manière concise l'ensemble des réalisations de surface possibles dans des langues offrant un plus grand degré de liberté sur l'ordre des syntagmes. Il est donc nécessaire, pour linéariser nos structures abstraites vers leur forme phonologique, de proposer des réalisations multiples, couvrant l'ensemble des combinaisons possibles. Afin de condenser ces descriptions, nous proposons dans ce chapitre une représentation intermédiaire, qui décrit des phrases modulo permutation de certains de leurs composants. Par suite, nous proposons plusieurs classes de langages s'appuyant sur cette représentation, et étudions la complexité algorithmique de plusieurs problèmes liés à l'analyse d'un énoncé selon ces représentations.

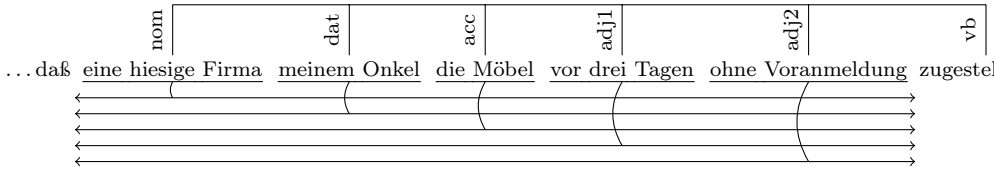
5.1 Motivation

La classe de phénomènes linguistiques que nous souhaitons modéliser, désignés par la suite sous le terme d'*ordres libres*, recouvre plusieurs notions. La plus évidente est la reconnaissance de phrases dans des langues à déclinaisons, où un marquage morphologique des cas (nominatif, accusatif, datif, *etc.*) est systématiquement présent et permet de retrouver la fonction syntaxique des arguments, indépendamment de leur position relative dans la phrase ; de telles langues, à l'image du latin, peuvent disposer d'un ordre canonique des arguments, mais autorisent fréquemment de larges déviations de cet ordre, sans affecter la grammaticalité ou le sens fondamental de la phrase [*cf.* Marouzeau, 1922].

Toutefois, le concept d'ordres libres peut également recouvrir des phénomènes plus restreints dans des langues imposant normalement des contraintes sur l'ordre de leurs arguments. Ainsi, les propositions subordonnées imbriquées en allemand sont usuellement construites de façon « parenthésée », en plaçant

les arguments nominaux du verbe au début, le verbe à la fin, et la proposition subordonnée entre les deux. Cependant, certains exemples attestent qu'il est possible, en préservant l'ordre des verbes à la fin, de mélanger librement leurs arguments, sans respecter le parenthésage induit par la structure abstraite. La reconstruction exacte de la structure syntaxique de ces énoncés (notamment les relations entre les verbes et leurs arguments) s'appuie alors usuellement sur des considérations sémantiques et pragmatiques.

Un premier exemple issu de Haider [1991] et repris dans Becker *et al.* [1992] est illustré par la figure 5.1. Dans cet exemple, les différents arguments de la proposition subordonnée forment des blocs, dont l'ordre relatif est quelconque : seule la position du verbe (rejeté à la fin) est constante.

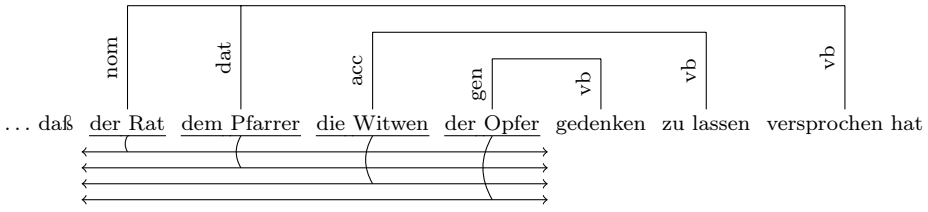


Exemple :

. . . daß vor drei Tagen meinem Onkel eine hiesige Firma ohne Voranmeldung die Möbel zugestellt hat.
. . . que il y a trois jours à mon oncle une entreprise locale sans préavis le mobilier a livré.
. . . qu'une entreprise locale a livré le mobilier à mon oncle sans préavis il y a trois jours.

FIGURE 5.1 – Liberté dans l'ordre des mots en allemand

Un second exemple, tiré de Becker *et al.* [1991] et donné dans la figure 5.2, illustre la possibilité de mélanger les arguments issus de plusieurs propositions subordonnées imbriquées. L'exemple comporte trois propositions subordonnées imbriquées, qui sont construites dans l'ordre canonique en haut de la figure : le placement de leurs arguments est toutefois libre, et indépendant de l'ordre (fixé) des verbes figurant à la fin, lequel permet de déduire la structure de la phrase. Observons que cette liberté permet de mélanger entre eux des composants appartenant à plusieurs propositions distinctes.



Exemple :

. . . daß die Witwen der Opfer dem Pfarrer der Rat gedengken zu lassen versprochen hat.
. . . que les veuves les victimes au prêtre le conseil commémorer de laisser a promis.
. . . que le conseil a promis au prêtre de laisser les veuves commémorer les victimes.

FIGURE 5.2 – Mélange des arguments en allemand (*scrambling*)

Notre objectif dans la suite de ce chapitre est de proposer une modélisation des phénomènes d'ordre libre, qui décrive de manière complètement

uniforme l'ensemble des ordres des mots autorisés par une langue ou par un phénomène donné. Cette modélisation exclut donc la possibilité de traiter des nuances de sens et d'emphase véhiculées par le choix d'un ordre en particulier de préférence à un autre. Tous les phénomènes offrant un degré modéré de liberté dans l'ordre des mots ne seront pas descriptibles par l'outil que nous proposons : ainsi, nous ne pourrons pas proposer de représentation unique pour toutes les réalisations d'une proposition indépendante en allemand, où le verbe doit obligatoirement occuper la seconde position, mais où le placement de ses arguments est libre. Choisi pour ses propriétés algorithmiques, cet outil permet seulement de rendre compte d'un placement séquentiel entre les syntagmes (comme dans une grammaire de réécriture classique) ou de leur permutation libre, c'est à dire sans contrainte aucune entre leurs positions respectives. En dépit de cette limitation, il semble néanmoins adéquat pour modéliser simplement la grammaticalité de langues comme le latin, ou pour factoriser partiellement des réalisations équivalentes dans des langues comme l'allemand.

5.2 L'algèbre $\mathrm{Comm}(\Sigma)$

Mots commutatifs Nous introduisons maintenant la notion de mots modulo permutations, ou *mots commutatifs* sur un alphabet, qui sera le support de notre représentation des ordres libres. Ces mots commutatifs sont formés soit comme des mots usuels, par la concaténation successive de leurs lettres ; soit comme des mots libres dont les composants peuvent être lus indifféremment dans n'importe quel ordre. L'ensemble des mots commutatifs sur un alphabet est défini à partir des lettres de cet alphabet, d'une opération binaire de concaténation, et d'une autre opération binaire de composition libre, qui permet d'assembler des mots commutatifs sans spécifier leur ordre de lecture.

Définition 5.1. L'ensemble $\mathrm{Comm}(\Sigma)$ des *mots commutatifs* construits sur un alphabet Σ est l'ensemble des termes de l'algèbre suivante :

- Le mot vide ε est un terme de $\mathrm{Comm}(\Sigma)$.
- Tout élément $l \in \Sigma$ est un terme de $\mathrm{Comm}(\Sigma)$.
- L'opération binaire $\odot$ (*concaténation*) est associative.
- L'opération binaire $\otimes$ (*composition libre*) est associative/commutative.

Les termes de $\mathrm{Comm}(\Sigma)$ peuvent représenter de façon abstraite des phrases, dans lesquelles l'ordre de certains composants est laissé libre. Remarquons que tout terme construit sans utiliser l'opération $\otimes$ s'interprète de façon transparente comme un mot de Σ^* : l'opération $\odot$ est identique à la concaténation usuelle.

Notion d'équivalence Les propriétés des opérations $\odot$ et $\otimes$ (associativité et commutativité) décrivent une notion d'équivalence entre les termes

de Comm(Σ), sur laquelle nous nous appuierons fortement par la suite. En particulier, deux termes sont équivalents si la seule chose qui les distingue est l'ordre des arguments d'une composition libre.

Définition 5.2. Deux termes $t, t' \in \mathrm{Comm}(\Sigma)$ sont dits immédiatement équivalents si et seulement si l'une de ces trois conditions est vérifiée :

— $t = C[\odot(\odot(t_1, t_2), t_3)]$ et $t' = C[\odot(t_1, \odot(t_2, t_3))]$ (associativité de $\odot$)
— $t = C[\otimes(\otimes(t_1, t_2), t_3)]$ et $t' = C[\otimes(t_1, \otimes(t_2, t_3))]$ (associativité de $\otimes$)
— $t = C[\otimes(t_1, t_2)]$ et $t' = C[\otimes(t_2, t_1)]$ (commutativité de $\odot$)

La figure 5.3 illustre cette définition.

Par extension, l'*équivalence* de t et t', notée $t \equiv_c t'$, est définie comme la clôture réflexive, symétrique et transitive de cette relation.

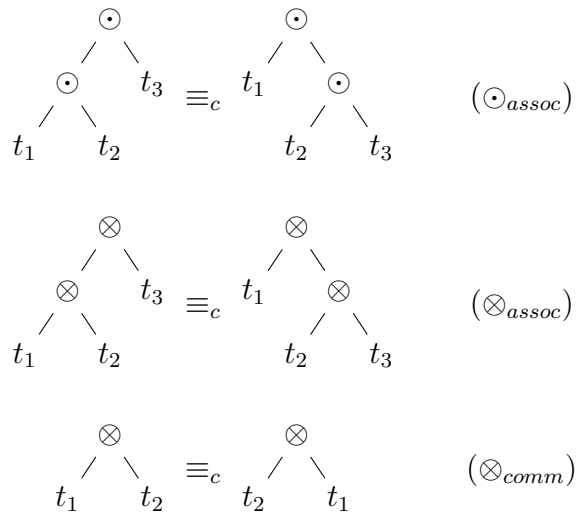


FIGURE 5.3 – Cas d'équivalence immédiate entre deux termes

Dans la suite, la notion d'équivalence issue de l'associativité de $\odot$ sera d'une importance réduite : l'équivalence entre $(a \odot b) \odot c$ et $a \odot (b \odot c)$, dénotant tous deux l'unique mot abc ne revêt pas d'importance particulière. En revanche, la combinaison de l'associativité et de la commutativité de $\otimes$ permettent de choisir librement l'ordre des composants d'un mot : ainsi, les termes $(a \otimes b) \otimes c$ et $a \otimes (c \otimes b)$ sont équivalents, matérialisant le fait que les mots abc et acb résultent tous deux de la composition libre des trois lettres a, b et c.

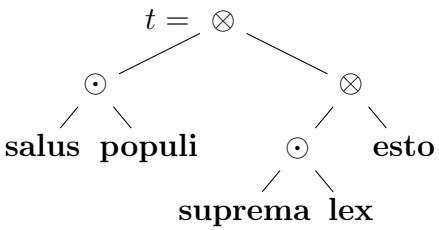
Langage d'un mot commutatif Nous définissons maintenant la notion de langage d'un terme de Comm(Σ) : il s'agit de l'ensemble des mots sur l'alphabet Σ pouvant être obtenus en choisissant un ordre linéaire quelconque pour les opérations de composition libre. Ces ordres peuvent être obtenus en faisant jouer les propriétés d'associativité et de commutativité de $\otimes$, puis en lisant simplement les feuilles des termes résultants de gauche à droite.

Définition 5.3. La *frontière* (ou *yield*) $\mathrm{frt}(t)$ d'un terme $t \in \mathrm{Comm}(\Sigma)$ est définie comme le mot de Σ^* obtenu en concaténant les feuilles de t de gauche à droite :

— $\mathrm{frt}(\varepsilon) = \varepsilon$
— $\mathrm{frt}(l) = l$ pour tout $l \in \Sigma$
— $\mathrm{frt}(\odot(t_1, t_2)) = \mathrm{frt}(t_1).\,\mathrm{frt}(t_2)$
— $\mathrm{frt}(\otimes(t_1, t_2)) = \mathrm{frt}(t_1).\,\mathrm{frt}(t_2)$

Le *langage* $\mathcal{L}(t)$ d'un terme t est défini comme l'ensemble des frontières des termes équivalents à t : $\mathcal{L}(t) = \{\mathrm{frt}(t') \mid t' \equiv_c t\}$.

Cette définition traduit la notion qu'un mot commutatif correspond à un ensemble de mots, formé de tous les choix possibles entre ses composants librement ordonnés. Une conséquence immédiate est que le langage d'un terme est fini, et que sa taille est bornée exponentiellement par la taille du terme dans le pire des cas. La figure 5.4 exemplifie la notion de langage d'un terme en listant les différentes permutations d'un terme représentant une phrase latine.



$$\mathcal{L}(t) = \left\{ \begin{array}{l} \text{Salus populi suprema lex esto.} \\ \text{Suprema lex salus populi esto.} \\ \text{Salus populi esto suprema lex.} \\ \text{Suprema lex esto salus populi.} \\ \text{Esto suprema lex salus populi.} \\ \text{Esto salus populi suprema lex.} \end{array} \right\}$$

FIGURE 5.4 – Terme représentant une phrase commutative et langage associé

Une autre propriété des termes de l'algèbre que nous avons définie pouvant en faciliter l'intuition est que le ré-ordonnancement des composants est local à un opérateur. En effet, les propriétés des opérateurs $\odot$ et $\otimes$ ne leur permettent pas d'interagir : les éléments d'un « groupe » combiné par $\otimes$ ne peuvent pas se mêler à ceux d'un autre groupe lorsque ces deux groupes sont séparés par l'opérateur $\odot$.

Une conséquence inattendue qui illustre ce principe est que le mot vide n'est *pas* un élément neutre pour $\mathrm{Comm}(\Sigma)$, comme illustré par les langages des termes de la figure 5.5. Sur la figure, les deux termes ont pour initialement pour frontière le mot *abc* : tous deux sont équivalents à un terme ayant

pour frontière *bac* par commutation de $\otimes(a, b)$, à un terme produisant *cab* par commutation de l'opérateur $\otimes$ situé à la racine, et incluent également dans leur langage le mot *cba* en commutant les deux occurrences de l'opérateur $\otimes$. Le premier terme t_1 est également équivalent à $\otimes(a, \otimes(b, c))$ par associativité de $\otimes$, permettant d'obtenir par commutation de $\otimes(b, c)$ la frontière *acb*, et le mot *bca* peut également être obtenu en commutant au préalable $\otimes(a, b)$. En revanche, le second terme t_2 n'inclut pas ces deux derniers mots dans son langage associé : la présence d'une opération de concaténation $\odot(x, \varepsilon)$ entre les deux occurrences de l'opérateur $\otimes$ interdit d'appliquer la règle d'associativité de ce dernier.

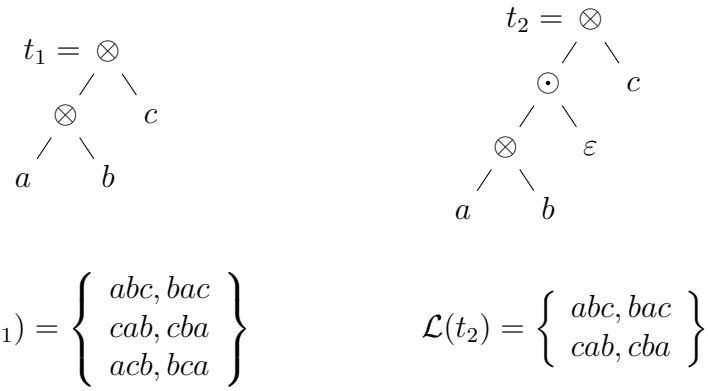


FIGURE 5.5 – Rupture de l'associativité de $\otimes$ par $\odot(\varepsilon, x)$

Alternativement, les mots commutatifs peuvent être envisagés comme des séquences (formées par $\odot$) ou des multi-ensembles [1] (formés par $\otimes$) de mots commutatifs plus petits. L'opérateur $\odot$ construit alors des séquences plus longues par concaténation de celles associées à ses arguments, et l'opérateur $\otimes$ construit des multi-ensembles comme l'union de ceux associés à ses arguments (un mot ou une lettre étant dans ce cas traité comme un singleton). Un multi-ensemble peut alors être réalisé comme un mot en concaténant tous ses éléments dans n'importe quel ordre.

5.3 Classes de grammaires pour Comm(Σ)

Les mots commutatifs décrits dans la section précédente constituent un outil de modélisation des énoncés contenant des composants librement ordonnés ; afin de modéliser des langues contenant de tels énoncés, nous avons maintenant besoin d'une notion de *grammaires commutatives*, produisant des langages de

1. Un multi-ensemble est un ensemble dont la fonction d'appartenance a pour domaine d'arrivée $\mathbb{N}$ au lieu de $\{0, 1\}$ [*cf.* Knuth, 1997, p. 694].

mots commutatifs. Étant donné que nos mots commutatifs sont définis comme des termes sur l'algèbre $\mathrm{Comm}(\Sigma)$, nous nous appuierons sur des grammaires de termes construisant des éléments de cette algèbre pour décrire des langages comportant des phénomènes d'ordres libres.

La classe de grammaires résultante manipule des tuples de contextes sur l'alphabet gradué $\{\odot_2, \otimes_2, \varepsilon_0\} \cup \Sigma_0$, à l'image des tuples de chaînes que manipulent les MCFG décrites section 2.4.1. Nous adoptons pour les productions de ces grammaires la notation décrite ci-dessous, similaire à celle de la définition 2.16.

Définition 5.4. Une *grammaire commutative* G est un tuple $(\Sigma, \mathcal{N}, S, \mathcal{P})$ où :
— Σ est l'*alphabet* de G.
— $\mathcal{N}$ est un ensemble de *symboles non-terminaux* typés.
— $S \in \mathcal{N}$ est un symbole non-terminal de type $[0]$ nommé *axiome*.
— $\mathcal{P}$ est un ensemble de *productions* construisant des tuples de contextes.

Les *types* des symboles non-terminaux (notés $\alpha, \beta, \dots$) sont des tuples (non-vides) de types simples construits à partir d'un unique type atomique 0 :
— 0 est un type simple (représentant un terme clos).
— Si τ et σ sont des types simples, $\sigma \to \tau$ est un type simple (contexte).
— Si $\tau_1 \dots \tau_n$ sont des types simples, $[\tau_1, \dots, \tau_n]$ est un type (tuple).

Les productions de $\mathcal{P}$ sont décrites au moyen de lambda-termes typés représentant des contextes, utilisant des tuples de variables issues du membre droit. Ainsi, toute production de $\mathcal{P}$ est de la forme :

$$A(M_1^{\alpha_1}, \dots M_n^{\alpha_n}) \leftarrow B_1(x_{1,1}, \dots, x_{1,n_1}) \dots B_p(x_{p,1}, \dots, x_{p,n_p})$$

où chaque M_i pour $i \in [n]$ est un lambda-terme linéaire simplement typé utilisant les constantes de $C = \{\odot^{0\to0\to0}, \otimes^{0\to0\to0}, \varepsilon^0\} \cup \{l^0 \mid l \in \Sigma\}$, et les variables libres $\mathrm{FV}(M_i) \subseteq \{x_{j,k} \mid j \in [p], k \in [n_j]\}$.

Toute production (ayant la forme décrite plus haut) doit respecter les types des non-terminaux de la façon suivante :
— Le non-terminal A a le type $[\alpha_1, \dots, \alpha_n]$.
— Chaque non-terminal B_j pour $j \in [p]$ est de type $[\alpha_{j,1}, \dots, \alpha_{j,n_j}]$.
— Toute variable $x_{j,k}$ est de type $\alpha_{j,k}$.

Une production dans laquelle l'une des variables $x_{i,j}$ est absente du membre gauche est dite *effaçante*. À l'inverse, une production dans laquelle une variable $x_{i,j}$ apparaît plusieurs fois dans le membre gauche est dite *parallèle*. Une grammaire commutative G contenant au moins une production effaçante (resp. parallèle) est également dite *effaçante* (resp. *parallèle*).

Dans la suite, nous désignerons fréquemment par la seule lettre $\mathcal{X}$ l'ensemble de variables $\{x_{j,k} \mid j \in [p], k \in [n_p]\}$ apparaissant dans une production de longueur p, dont les symboles non-terminaux sont d'arités respectives n_p. En outre, nos démonstrations portant sur les productions emploieront par convention la variable i pour désigner un indice allant de 1 à n (itérant sur le tuple

associé au membre gauche d'une production) et les variables j et k pour des indices appartenant respectivement à $[p]$ et $[n_j]$ (permettant de considérer l'ensemble des variables $x_{j,k}$ apparaissant dans une production). Enfin, sauf mention contraire, les productions et grammaires commutatives présentes dans ce chapitre ne seront ni effaçantes, ni parallèles.

Toute grammaire commutative définit un langage de termes de la même manière qu'une MCFG définit un langage de mots ; la contrainte sur le type de l'axiome (à savoir $[0]$) garantissant que les éléments de son langage représentent un unique terme clos. Par suite, le langage de mots d'une grammaire commutative est défini naturellement comme l'union des langages des termes qu'elle génère.

Définition 5.5. Étant donné une grammaire commutative $G = (\Sigma, \mathcal{N}, S, \mathcal{P})$, le langage de termes $\mathcal{T}_G(A)$ selon G d'un symbole non-terminal $A \in \mathcal{N}$ de type $[\alpha_1, \ldots, \alpha_n]$ est l'ensemble des tuples de lambda-termes $(P_1^{\alpha_1}, \ldots, P_n^{\alpha_n})$ respectant les conditions suivantes :

— $A(M_1, \ldots, M_n) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}) \ldots B_p(x_{p,1}, \ldots, x_{p,n_p})$ est dans $\mathcal{P}$.
— Pour tout $j \in [p]$, $\left(N_{j,1}, \ldots, N_{j,n_j}\right)$ appartient à $\mathcal{L}(B_j)$.
— Pour tout $i \in [n]$, $j \in [p]$ et $k \in [n_j]$, $P_i = M_i[x_{j,k} \leftarrow N_{j,k}]$.

Le *langage de termes* $\mathcal{T}(G)$ de G est alors défini comme celui de son axiome $\mathcal{T}_G(S)$. Par extension, le *langage* (de mots) $\mathcal{L}(G)$ de G est défini comme l'union des langages des termes appartenant à son langage de termes :

$$\mathcal{L}(G) = \bigcup_{t \in \mathcal{T}(G)} \mathcal{L}(t)$$

Nous considérons plusieurs classes de grammaires commutatives, comme candidats potentiels dans le but de capturer l'ensemble des langages naturels dotés de phénomènes d'ordres libres. Ces classes étendent aux ordres libres la hiérarchie des grammaires catégorielles abstraites du second ordre ($\mathbf{ACG}(2, m)$) sur les mots décrite dans la section 2.4.2, laquelle inclut les grammaires hors-contexte multiples (MCFG), hors-contexte (CFG) et régulières (RG). Du point de vue du traitement des langues, ces classes peuvent être considérées comme une tentative d'étendre aux ordres libres de la notion de grammaire légèrement sensible au contexte.

Comme pour les MCFG, nous obtenons nos différentes classes de grammaire en restreignant le type des symboles non-terminaux, et éventuellement la forme des productions.

Définition 5.6. Nous introduisons plusieurs classes de grammaires commutatives, définies par les contraintes suivantes :

— Une *grammaire hors contexte multiple commutative* (CMCFG) est une grammaire commutative non-restreinte. Ces grammaires associent des tuples de contextes à leurs symboles non-terminaux.

— Une *grammaire régulière multiple commutative* (CMREG) est une grammaire commutative dont tous les symboles non-terminaux sont de type $[\alpha_1, \ldots, \alpha_n]$, avec $\alpha_1 = \cdots = \alpha_n = 0$. Ces grammaires associent des tuples de termes clos à leurs symboles non-terminaux.
— Une *grammaire à macros commutative* (CMG) est une grammaire commutative dont tous les symboles non-terminaux sont de type $[\alpha]$. Ces grammaires associent un unique contexte à chacun de leurs symboles non-terminaux.
— Une *grammaire hors contexte commutative* (CCFG) est une grammaire commutative dont tous les symboles non-terminaux sont de type $[0]$. Ces grammaires associent un unique terme clos à chaque symbole non-terminal.
— Une *grammaire régulière commutative* (CRG) est une grammaire commutative dont les symboles non-terminaux sont de type $[0]$, et dont toutes les productions associent à leur membre gauche un contexte linéaire droit – c'est-à-dire un terme clos dont tous les fils gauches sont des feuilles constantes (lettres de Σ ou ε), et dont le dernier fils droit est l'unique variable, $x_{1,1}$.

À l'exception des CMREG, nos classes de grammaires commutatives sont nommées d'après les classes de grammaires de mots usuelles qu'elles généralisent (voir section 5.3.2). Le nom de la classe des CMREG provient pour sa part de la classe de grammaires de termes qui la sous-tend, les grammaires d'arbres régulières multiples [Mönnich, 2007].

La définition de nos différentes classes de grammaires commutatives induit immédiatement une hiérarchie, représentée par la figure 5.6. Les flèches représentent les relations d'inclusion entre les différentes classes.

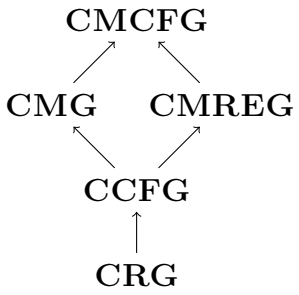


FIGURE 5.6 – Hiérarchie des grammaires commutatives

Illustration Nous illustrons maintenant le fonctionnement de nos grammaires, au travers de deux courts exemples.

Soit $G_1 = (\Sigma, \mathcal{N}_1, S, \mathcal{P}_1)$ la grammaire définie sur l'alphabet $\Sigma = \{a, b, c\}$ au moyen des non-terminaux suivants : $\mathcal{N}_1 = \left\{S^{[0]}, A^{[0\to0]}, B^{[0\to0]}, C^{[0\to0]}\right\}$. Les productions de $\mathcal{P}_1$ sont les suivantes :

1. $S(x\ \varepsilon) \leftarrow A(x)$
2. $A(\lambda t^0.\ \odot\ a\ (x\ t)) \leftarrow B(x)$
3. $B(\lambda t^0.x\ (\odot\ b\ t)) \leftarrow C(x)$
4. $C(\lambda t^0.\ \odot\ (x\ t)\ c) \leftarrow A(x)$
5. $A(\lambda t^0.t) \leftarrow$

Cette grammaire construit au moyen de ses non-terminaux A, B et C un contexte unaire, ayant la forme donnée par la figure 5.7. La production 5 (terminale) permet de partir d'un contexte trivial, et les productions 2 à 4 ajoutent respectivement un a concaténé à gauche au dessus du contexte x, un c concaténé à droite au dessus de x, et un b concaténé en dessous de x. Enfin, la première production complète un contexte produit par A au moyen du symbole ε (qui se substitue à t^0 dans la figure).

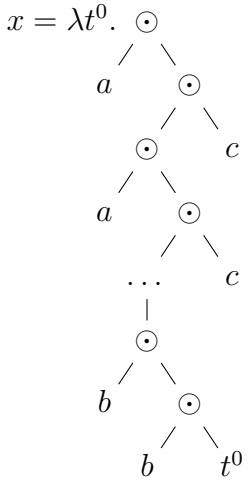


FIGURE 5.7 – Forme des contextes produits par le non-terminal A de G_1

Les termes construits par G_1 ne contenant que le symbole $\odot$, le langage de mots $\mathcal{L}(G_1)$ qui lui est associé s'obtient immédiatement en considérant l'ensemble des frontières des éléments de son langage de termes. Ceux-ci ayant tous la forme illustrée par la figure 5.7, leur frontière est un mot contenant n occurrences de la lettre a à gauche, n occurrences de c à droite, et n occurrences de b entre les deux ; par conséquent : $\mathcal{L}(G_1) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Considérons maintenant une seconde grammaire $G_2 = (\Sigma, \mathcal{N}_2, S, \mathcal{P}_2)$ qui soit construite sur le même alphabet, avec $\mathcal{N}_2 = \left\{S^{[0]}, A^{[0,0,0]}\right\}$, et utilisant les productions suivantes :

1. $S(\otimes\ (\otimes\ x_1\ x_2)\ x_3) \leftarrow A(x_1, x_2, x_3)$
2. $A(\otimes\ a\ x_1, \otimes\ b\ x_2, \otimes\ c\ x_3) \leftarrow A(x_1, x_2, x_3)$
3. $A(\varepsilon, \varepsilon, \varepsilon) \leftarrow$

Le langage du non-terminal A est formé par des triplets de termes, combinant un nombre égal d'occurrences des symboles a, b et c (respectivement) au moyen de l'opérateur $\otimes$; l'axiome S combine à son tour les termes résultants au moyen du même opérateur. Ainsi, les termes de $\mathcal{T}(G_2)$ ont pour frontière un mot $a^n b^n c^n$, et tous leurs nœuds internes sont formés par l'opérateur $\otimes$. Les règles d'équivalence et la notion de langage d'un terme données par les définitions 5.2 et 5.3 permettent alors de réordonner librement les feuilles des termes générés par G_2 ; par conséquent, $\mathcal{L}(G_2) = \{w \mid |w|_a = |w|_b = |w|_c\}$.

Enfin, observons que les types des non-terminaux de G_1 sont des tuples de taille 1 (contenant soit le type atomique 0, soit le type d'un contexte unaire $0 \to 0$) ; par conséquent $G_1 \in \mathbf{CMG}$. En ce qui concerne G_2, les types de ses non-terminaux sont des tuples de taille variable, contenant uniquement le type atomique 0 ; par conséquent, $G_2 \in \mathbf{CMREG}$.

5.3.1 Semilinéarité

Une propriété essentielle des langages issus des grammaires commutatives que nous décrivons est leur semilinéarité. Cette propriété traduit le fait que le nombre de lettres dans les mots d'un tel langage respecte certaines proportions.

Définition 5.7. Un langage L est dit *semilinéaire* si son image de Parikh (formée par l'ensemble des images de Parikh de ses mots, voir section 2.1.2) constitue un ensemble semilinéaire (définis section 2.1.4).

La semilinéarité d'un langage est directement liée à la notion de *croissance constante* évoquée par Joshi [1985], qui est une propriété essentielle des langages légèrement sensibles au contexte[Joshi *et al.*, 1990]. Bien que ces deux propriétés ne soient pas exactement équivalentes (voir Kallmeyer [2010]), la première implique la seconde, et capture également l'intuition qui a présidé à sa formulation. Les CFG (et plus généralement les MCFG) produisent des langages semilinéaires, et la semilinéarité de nos langages de mots commutatifs peut être montrée par un argument de pompage exactement similaire à celui établissant la semilinéarité des langages hors-contexte, montrée originellement par Parikh [1966]. Une preuve claire et accessible suivant ce schéma pour les CFG peut se trouver dans Kozen [1997].

5.3.2 Généralisation des grammaires de mots

La hiérarchie que nous proposons pour les grammaires commutatives reflète la hiérarchie existante des MCFG sur les chaînes évoquée dans la section 2.4.1 ; et elle inclut successivement les classes de langages décrites par ces grammaires. Plus précisément, les langages décrits par nos grammaires coïncident avec les classes de langages de mots usuelles en interdisant l'emploi du symbole $\otimes$ dans les productions. Ce résultat est immédiatement apparent pour les classes de

grammaires commutatives manipulant des termes clos : un terme t combinant des lettres à l'aide du seul opérateur de concaténation $\odot$ équivaut immédiatement au mot obtenu en lisant sa frontière $\mathrm{frt}(t)$.

Plus généralement, cette correspondance peut être établie en exploitant des résultats existants dans la littérature des ACG : il est possible de construire une ACG d'ordre 2 sur les chaînes à partir d'une CMCFG sans $\otimes$, qui reconnaisse le même langage. Cette ACG peut successivement être simulée par un transducteur déterministe cheminant dans un arbre (DTWT) comme montré par Salvati [2007], puis par un système de réécriture linéaire hors-contexte (LCFRS), et donc par une MCFG, d'après Weir [1992].

Nous esquissons brièvement la technique permettant de construire une 2-ACG à partir d'une CMCFG. Cette construction exploite notre choix de représenter nos grammaires d'une manière similaire aux MCFG à l'aide du lambda-calcul, et reprend la méthode usuelle permettant de simuler un formalisme hors-contexte par une ACG, détaillée dans de Groote et Pogodalla [2004]. Pour rappel, la définition d'une ACG est donnée en 2.18. Soit $G = (\Sigma, \mathcal{N}, S, \mathcal{P})$ une CMCFG, nous construisons une ACG $G' = (\Sigma_a, \Sigma_c, \Lambda, S)$ dans laquelle :

- Le vocabulaire abstrait Σ_a représente les dérivations de G de la manière suivante : l'ensemble des types atomiques de Σ_a est formé par l'ensemble $\mathcal{N}$ des non-terminaux de G, et l'ensemble des constantes est formé par l'ensemble $\mathcal{P}$ de ses productions. Le type des constantes est donné par la structure des productions, c'est-à-dire que la constante p correspondant à une production de la forme $A(\dots) \leftarrow B_1(\dots)\dots B_n(\dots)$ a pour type $\tau_p = B_1 \to \cdots \to B_n \to A$. Les termes en forme normale construits sur Σ_a représentent ainsi des arbres de dérivation selon G, le typage des constantes garantissant leur bonne formation.
- Le vocabulaire concret Σ_c représente des mots sur Σ^* de la manière suivante : son unique type atomique est $*$, et ses constantes sont les lettres de Σ (ou ε), toutes munies du type $* \to *$. Cette représentation suit le schéma décrit dans la section 2.2.4.
- Le lexique Λ associe à chaque production un lambda-terme représentant son membre gauche, utilisant éventuellement la représentation des tuples et des projections évoquée par la section 2.2.4. L'image d'une feuille a^0 est la constante $a^{*\to *}$ associée à la lettre de Σ correspondante, et l'image de la constante $\odot^{0\to 0\to 0}$ est directement remplacée par le terme $\lambda f g x.f\ (g\ x)$ de type $(* \to *) \to (* \to *) \to * \to *$ dénotant la concaténation. Le terme du langage concret associé à une dérivation valide se normalise ainsi en une représentation de l'unique chaîne associée au terme de Comm(Σ) dérivé.
- Le type distingué S de G' est le type atomique associé à l'axiome de G.

Observons que dans le cas d'une CMG, il n'est pas nécessaire de représenter les tuples associés aux non-terminaux (puisque ceux-ci sont tous de taille 1). La construction précédente permet alors à une CMG d'être représentée par une

ACG munie d'un lexique de complexité 3, une CCFG employant un lexique de complexité 2 et une CRG correspondant à un lexique de complexité 1. Ce résultat reflète la hiérarchie des ACG mise en valeur dans la section 2.4.2.

Nous récapitulons dans la figure 5.8 la hiérarchie des classes de *langages* issues des différentes classes de grammaires commutatives : une flèche simple représente une relation d'inclusion stricte, et les flèches doubles décorées par « ? » dénotent une relation d'inclusion ou d'égalité ; l'abréviation « … L » désigne la classe de langages $\mathcal{L}(\ldots \mathbf{G})$. Les résultats d'inégalité entre les différentes classes découlent de résultats de difficulté algorithmique présentés dans la prochaine section. Pour référence, les classes usuelles de langages légèrement sensibles au contexte figurent également sur le schéma, reliées aux classes de langages de mots commutatifs qui les généralisent par un trait simple.

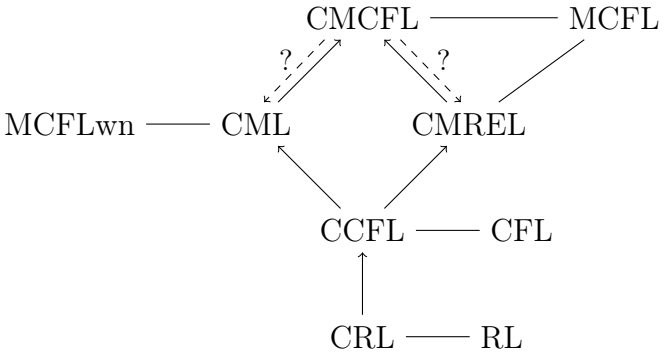


FIGURE 5.8 – Hiérarchie des langages de mots commutatifs

5.4 Problème de l'analyse

Une fois proposées les classes de grammaires permettant de décrire des langages de mots commutatifs, nous souhaitons étudier plus avant le problème de l'analyse. En raison de l'étape intermédiaire due à la notion de langage d'un terme commutatif, la relation entre un arbre de dérivation selon une grammaire commutative et le mot qu'il reconnaît n'est pas complètement triviale ; elle doit tenir compte de l'associativité et de la commutativité de l'opérateur $\otimes$. Toutefois, cette étape intermédiaire peut-être simplifiée : intuitivement, les mots commutatifs combinés par cet opérateur ont simplement besoin d'être comptés. Par exemple, décider si un mot appartient au langage de la grammaire G_2 décrite page 123 revient simplement à compter les occurrences de a, b et c, ce qui est algorithmiquement simple : de fait, il existe une grammaire commutative régulière décrivant le même langage, et pour laquelle, comme nous allons le voir, l'analyse d'un mot w ne requiert pas d'établir explicitement l'équivalence entre le terme dérivé par la grammaire et un terme t ayant pour frontière $\mathrm{frt}(t) = w$.

Analyse et analyse universelle Nous étudions les deux variantes suivantes du problème de décision lié à l'analyse :

— Étant donné un mot w et une grammaire G, est-ce que $w \in \mathcal{L}(G)$? (Analyse universelle.)

— Étant donné un mot w, est-ce que $w \in \mathcal{L}(G)$? (Analyse selon une grammaire G fixée à l'avance.)

La différence entre ces deux variantes tient dans la complexité algorithmique de la procédure : l'analyse universelle dépend de la taille de la grammaire G (et du mot w), alors que l'analyse selon une grammaire fixée ne dépend que de w, la taille de la grammaire étant considérée comme une constante de l'algorithme. Cette différence trouve son importance dans des applications liées au traitement automatique des langues. D'une part, beaucoup d'applications pratiques requièrent l'analyse de nombreux énoncés selon une grammaire fixe, offrant une certaine latitude pour pré-compiler ou optimiser l'analyseur correspondant. D'autre part, il est envisageable pour des raisons psycholinguistiques que nombre de langages naturels comprenant des phénomènes d'ordres libres ne requièrent pas en pratique toute la complexité atteignable par notre formalisme dans le pire cas : de tels langages pourraient éventuellement être décrits une grammaire commutative relativement peu coûteuse par rapport à d'autres grammaires de sa classe.

Résultats du chapitre La table 5.1 résume l'ensemble des résultats de complexité établis dans ce chapitre, incluant le cas des grammaires avec effacement. Pour chaque classe, le suffixe « -C » désigne un problème complet pour la classe en question et « -D » un problème difficile pour cette classe.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-C	
CCFG	LogCFL-C	NP-C	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE-C	EXPTIME-C
CMCFG	NP-C	PSPACE-D	EXPTIME-C

Table 5.1 – Récapitulatif des résultats de complexité pour l'analyse

La plupart des algorithmes d'analyse établissant ces résultats s'appuient sur le système de réécriture $\rightarrow_\varepsilon$ introduit dans la section 5.6, qui permet de compresser les termes commutatifs ; ces algorithmes sont donc détaillés dans la dernière section du chapitre. L'analyse à grammaire fixée des CRG et CCFG, qui est polynomiale, fait l'objet de deux algorithmes dédiés, détaillés dans la section 5.5. Enfin, cette section ci établit les bornes inférieures sur la complexité de l'analyse. Certaines sont issues de la littérature des MCFG, que nos

grammaires peuvent émuler directement à l'aide de l'opérateur $\odot$; d'autres sont propres à l'emploi de l'opérateur commutatif $\otimes$. Nous établissons également la NP-complétude de l'analyse universelle par un terme commutatif, consistant à déterminer si un mot w appartient à $\mathcal{L}(t)$ étant donné w et t.

5.4.1 Résultats issus des MCFG

La classe de complexité LogCFL caractérise l'ensemble des problèmes réductibles en espace logarithmique à la reconnaissance d'un langage hors-contexte. Par conséquent, l'analyse d'un mot selon une CCFG fixée (capable de simuler directement une CFG, comme mentionné dans la section 5.3.2) constitue un problème LogCFL-difficile.

Suivant le même raisonnement, l'analyse universelle pour les CMG et CMREG (capables de simuler une MCFG_{WN}) est PSPACE-difficile dans le cas des grammaires sans effacement. Ces résultats de complexité sur les MCFG, établis par Kaji *et al.* [1992], ont également inspiré le fonctionnement de l'algorithme d'analyse général décrit à la fin de ce chapitre, qui établit la complexité maximale de l'analyse universelle pour les CMG, CMREG et CMCFG. Enfin, l'analyse universelle selon des CMREG (et donc CMCFG) incluant des productions effaçantes (et donc capables de simuler une MCFG avec effacement) est, pour les mêmes raisons, EXPTIME-difficile ; ce résultat ne s'étend toutefois pas nécessairement aux CMG, dont la complexité est liée aux grammaires sans entrelacement (MCFG_{WN}).

La table 5.2 reprend le récapitulatif des résultats de complexité du chapitre, en mettant en valeur ceux qui sont issus des MCFG.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-C	
CCFG	LogCFL-D	NP-C	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE-D	EXPTIME-D
CMCFG	NP-C	PSPACE-D	EXPTIME-D

Table 5.2 – Résultats de complexité connus, issus des MCFG

5.4.2 Appartenance au langage d'un terme

Nous considérons maintenant le problème de l'appartenance d'un mot w au langage d'un terme commutatif t : ce problème est NP-complet. Nous décrivons dans cette sous-section un algorithme non-déterministe, s'exécutant en

temps polynomial, capable de décider si $w \in \mathcal{L}(t)$. La NP-difficulté de l'appartenance d'un mot au langage d'un terme commutatif sera, pour sa part, établie par réduction du problème (NP-complet) de 3-partition. Une conséquence immédiate de cette NP-difficulté est que l'analyse universelle selon une CCFG est également NP-difficile : pour tout terme t, il est possible de construire directement une CCFG dont l'axiome se réécrit en t, et dont le langage est donc exactement $\mathcal{L}(t)$.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-D	
CRG	NL	NP-C	
CCFG	LogCFL-C	NP-D	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE-C	EXPTIME-C
CMCFG	NP-C	PSPACE-D	EXPTIME-C

Table 5.3 – Résultats de complexité de la sous-section 5.4.2

Reconnaissance d'un mot selon un terme commutatif Soit w un mot de Σ^* et t un terme commutatif appartenant à $\mathrm{Comm}(\Sigma)$; par définition, s'il est possible de deviner un terme t' tel que $t \equiv_c t'$ et $\mathrm{frt}(t') = w$, alors $w \in \mathcal{L}(t)$. Il en résulte un algorithme non-déterministe, qui s'exécute en temps polynomial, puisque $|t| = |t'|$; la frontière w de t' peut être vérifiée immédiatement et l'équivalence entre t et t' peut être établie en fusionnant les occurrences voisines de chaque opérateur (aplatissant le terme), et en comparant les arbres résultants de manière descendante, modulo permutation des branches dominées par $\otimes$. La figure 5.9 illustre ce procédé, la permutation $\pi(1,2,3) = (2,3,1)$ permettant de choisir quels sont les sous-arbres à comparer deux à deux.

Problème de 3-partition (3-PART) Nous donnons d'abord une définition de 3-PART, qui est un problème NP-complet :

Instance Un ensemble $S = \{n_1 \dots n_{3m}\}$ d'entiers, et une constante k telle que $\frac{k}{4} < n < \frac{k}{2}$ pour tout $n \in S$, et que :

$$\sum_{n \in S} n = k.m$$

Solution *Vrai* si et seulement si il existe une partition $S_1 \dots S_m$ de S telle que chaque ensemble S_i respecte :

$$\sum_{n \in S_i} n = k$$

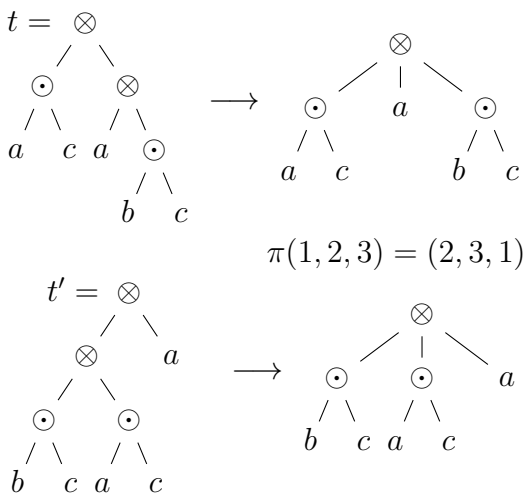


FIGURE 5.9 – Vérification de l'équivalence entre deux termes

Les contraintes initiales sur les n_i entraînent que tous les ensembles S_i d'une éventuelle solution contiennent exactement 3 éléments, notés $n_{i,1}$, $n_{i,2}$ et $n_{i,3}$. La figure 5.10 propose une représentation visuelle d'une instance de 3-PART et de sa solution, consistant en un ré-arrangement des éléments de S par groupes de 3, chaque ligne de la figure du bas correspondant à un des S_i.

Théorème 5.1 (Garey et Johnson [1990])**.** *Le problème* 3-PART *est* NP*-complet.*

Théorème 5.2. *L'appartenance d'un mot w au langage d'un terme commutatif t est un problème* NP*-complet.*

Démonstration. L'algorithme non-déterministe polynomial proposé plus haut permet de décider si $w \in \mathcal{L}(t)$, entraînant l'appartenance du problème à NP.

La preuve de NP-difficulté procède par réduction de 3-PART. Étant donné une instance $I = (S, k)$ de 3-PART nous construisons un terme t et un mot w tels que $w \in \mathcal{L}(t)$ si et seulement si I a une solution ; cette construction est illustrée par la figure 5.11. Soit $\Sigma = \{a, \#\}$ l'alphabet sur lequel sont construits t et w. Nous posons $w = (a^k\#)^m$, formant le mot à reconnaître en séparant m séquences de a, chacune de longueur k, à l'aide du symbole $\#$. Nous construisons pour tout $n \in S$ les termes $t_n = \odot(a, x)^n[\varepsilon]$ concaténant n fois la lettre a ; ainsi que le terme $t_\# = \otimes(\#, x)^m[\varepsilon]$ combinant librement m fois le symbole $\#$. Nous considérons alors le terme $t = \otimes(t_{n_1}, \cdots \otimes (t_{n_{3m}}, t_\#)\dots)$ combinant librement chacun des termes t_n et le terme $t_\#$, et montrons que $w \in \mathcal{L}(t) \Leftrightarrow$ **3-PART**$(I) = \mathit{Vrai}$.

Nous montrons d'abord la réciproque de l'implication. Si l'instance I a une solution $S_1 \dots S_m$, alors il existe m triplets d'entiers dans S dont la somme est égale à k. À chacun de ces triplets correspondent trois sous-termes de

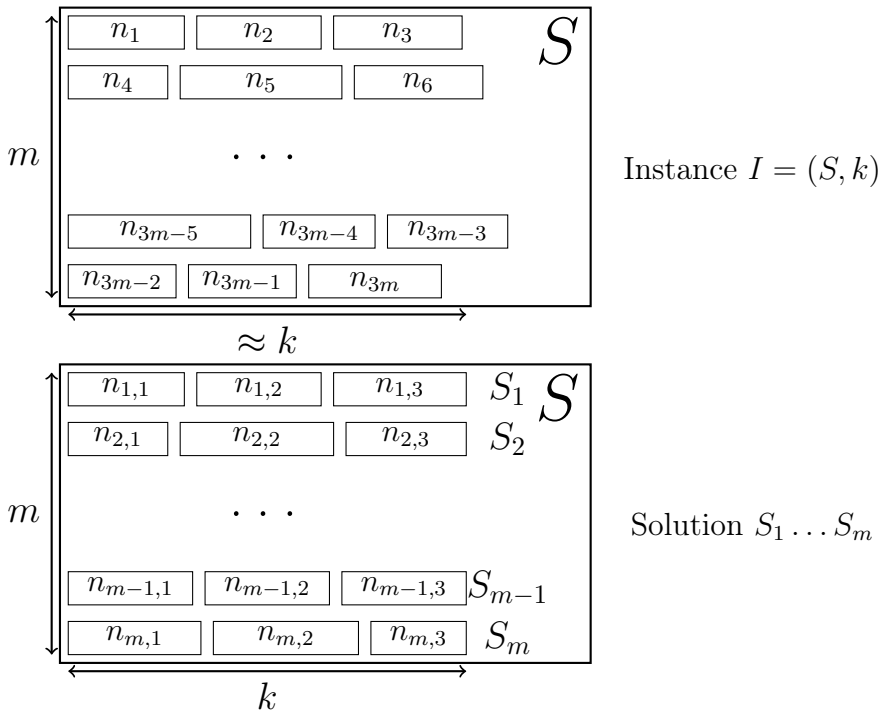


FIGURE 5.10 – Instance et solution associée de 3-PART

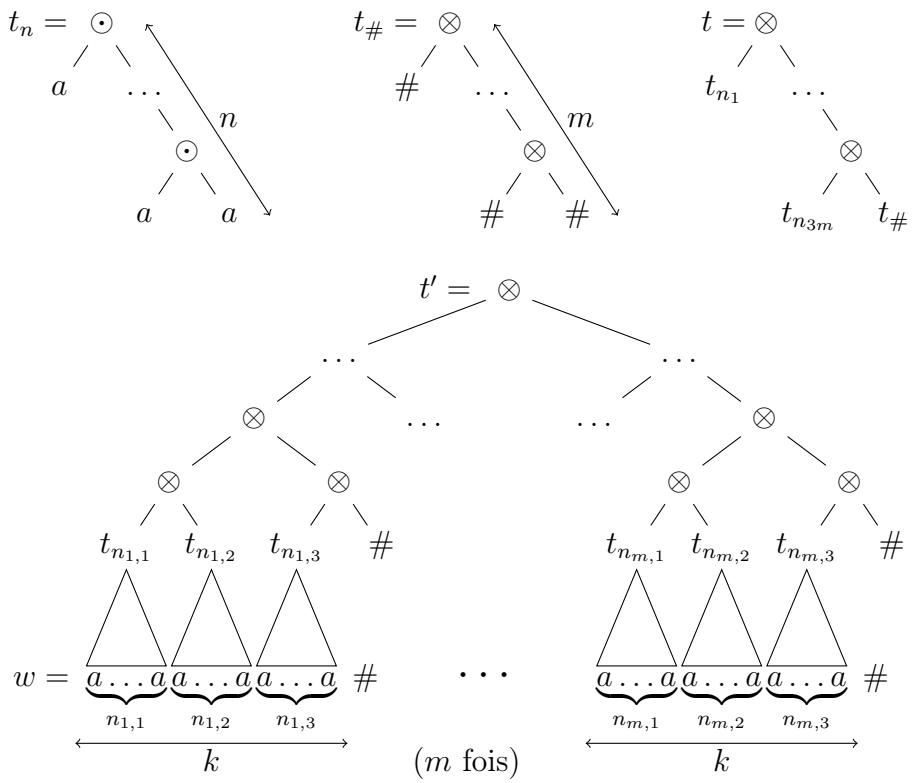


FIGURE 5.11 – Réduction d'une instance de 3-PART à $w \in \mathcal{L}(t)$

t, combinés au moyen de l'opérateur $\otimes$. Nous pouvons donc faire jouer les propriétés d'associativité et de commutativité de ce dernier pour regrouper ces triplets et les faire suivre d'un des m symboles # ajoutés par le sous-terme $t_{\#}$, formant ainsi un terme t' équivalent à t tel qu'illustré par la figure 5.11. La frontière de t' est alors exactement w, montrant que $w \in \mathcal{L}(t)$.

Nous montrons maintenant l'implication directe. Si $w \in \mathcal{L}(t)$, alors il existe un terme $t' \equiv_c t$ tel que $\mathrm{frt}(t') = w$. L'application des règles d'équivalence portant sur l'opérateur $\odot$ ne permet pas de de mêler les nœuds appartenant aux sous-termes t_n avec les autres, ni de modifier leur frontière. Le mot w étant la frontière de t', il contient donc nécessairement un facteur distinct a^n pour chaque $n \in S$. Or, w se compose par construction de m facteurs identiques de la forme $a^k\#$; par conséquent, nous pouvons déduire de la structure de t' une partition des sous-termes t_n dans laquelle chaque sous-ensemble contient k occurrences du symbole a. Cette partition se traduit directement en une partition de l'ensemble S dans laquelle la somme des éléments de chaque sous-ensemble est k, ce qui constitue une solution pour l'instance I. □

5.4.3 Résultats de difficulté supplémentaires

Nous établissons maintenant trois résultats de NP-difficulté, portant respectivement sur l'analyse universelle selon une CRG, et sur l'analyse à grammaire fixée selon une CMG et une CMREG. Ces bornes inférieures de complexité sont en fait optimales, mais l'existence d'algorithmes dans NP pouvant résoudre ces problèmes sera montrée à l'issue de ce chapitre, dans la section 5.6.3, leur fonctionnement requérant une notion de « compression » des termes qui en élimine les composantes inutiles à la formation du mot w.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-D	
CCFG	LogCFL-C	NP-D	
CMG	NP-D	PSPACE-D	EXPTIME
CMREG	NP-D	PSPACE-C	EXPTIME-C
CMCFG	NP-D	PSPACE-D	EXPTIME-C

Table 5.4 – Résultats de complexité de la sous-section 5.4.3

La NP-difficulté de l'analyse selon une CMG et une CMREG est établie, dans les deux cas, par une réduction à 3-PART similaire à celle décrite plus haut. Ces résultats entraînent immédiatement que l'analyse selon une CMCFG fixée est NP-difficile. Le cas de l'analyse universelle selon une CRG est, quant à lui, établi par réduction au problème de 3-couverture exacte, également NP-complet. Observons toutefois que la preuve de ce ce dernier résultat repose

crucialement sur la taille de l'alphabet Σ sur lequel repose la grammaire, et n'est plus valide si le mot w à analyser est construit sur un alphabet de taille constante. De fait, la complexité de l'analyse universelle à alphabet fixé est NL, identique à celle de l'analyse à grammaire fixée établie dans la prochaine section.

La construction de ces réductions, ainsi que leurs preuves détaillées, sont données en annexe dans la section B.1.

5.5 Algorithmes d'analyse pour les cas restreints

Nous présentons maintenant deux algorithmes d'analyse à grammaire fixée, destinés respectivement aux CRG et aux CCFG. Nous montrerons la correction et la complétude, ainsi que la complexité de ces algorithmes, qui demeure polynomiale dans le pire cas. En particulier, la complexité de l'analyse pour une CCFG fixée est la même que pour l'analyse d'une CFG.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-C	
CCFG	LogCFL	NP-C	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE-C	EXPTIME-C
CMCFG	NP-C	PSPACE-D	EXPTIME-C

Table 5.5 – Résultats de complexité de la section 5.5

Ces algorithmes alternent entre deux stratégies pour analyser des fragments d'un mot commutatif. Intuitivement, une série de fragments consécutifs d'un mot décrit par une grammaire commutative peut avoir été introduite de deux façons. Le premier cas est celui où le terme dérivé par la grammaire combine une série de symboles à l'aide de l'opérateur de concaténation $\odot$: dans ce cas, l'analyse procède de manière similaire à celle des grammaires hors contexte. Le second cas est celui où les composants ont pu être réordonnés, et descendent donc d'occurrences de l'opérateur $\otimes$ voisines dans le terme dérivé : dans ce cas, il s'agit d'un problème de comptage – le nombre de composants doit alors être cohérent avec ce que permettent les productions commutatives de la grammaire (à savoir celles utilisant uniquement l'opérateur $\otimes$) ; ce problème revient à décider de l'appartenance d'un vecteur à un ensemble semilinéaire.

5.5.1 Notions supplémentaires

Afin de simplifier la description des algorithmes d'analyse, nous introduisons deux notions intermédiaires : la première est celle de forme normale pour les CCFG, similaire à la notion de forme normale usuelle des grammaires de termes ou à la celle de forme normale de Chomsky pour les grammaires hors-contexte ; la seconde est celle de langage commutatif d'un symbole non-terminal, représentant l'ensemble des termes dérivables à partir d'un symbole non-terminal utilisant exclusivement l'opérateur $\otimes$.

En outre, par souci de simplification, nous exploitons librement dans cette section le fait que, dans une CCFG, tout symbole non-terminal dans une dérivation est associé à un unique terme clos. Nous parlerons ainsi du *langage étendu* (ou *langage non-terminal*) de termes d'un symbole non-terminal A selon une grammaire $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$, formé de l'ensemble des termes commutatifs sur $\mathrm{Comm}(\Sigma \cup \mathcal{N})$ dérivables à partir de A en utilisant les productions de $\mathcal{P}$. Ainsi, si $A(\otimes\ x\ y) \leftarrow B(x)\ C(y)$ est une production de $\mathcal{P}$, alors $\otimes\ B\ C$ appartient au langage de termes étendu de A ; par extension, les mots BC et CB formés sur $(\Sigma \cup \mathcal{N})^*$ appartiennent au langage (de mots) étendu de A.

Forme normale d'une CCFG Une grammaire hors contexte commutative en forme normale utilise exclusivement les formes suivantes dans ses productions :

$$\begin{array}{lll} (1) & A(op\ x\ y) \leftarrow B(x)\ C(y) & \\ (2) & A(op\ a\ x) \leftarrow B(x) & \text{avec } \begin{cases} op \in \{\odot, \otimes\} \\ a \in \Sigma \cup \{\varepsilon\} \end{cases} \\ (3) & A(a) \leftarrow & \end{array}$$

Une CCFG (resp. une CRG) est dite en *forme normale* si et seulement si toutes ses productions sont de la forme 1 ou 3 (resp. 2 ou 3). Pour toute CCFG ou CRG G, il existe une grammaire équivalente G' en forme normale, telle que $\mathcal{T}(G') = \mathcal{T}(G)$. Nous supposons dans le reste de cette section, sans perte de généralité, que toutes les grammaires que nous manipulons sont en forme normale.

Langage commutatif d'un symbole non-terminal Le langage commutatif d'un symbole non-terminal est l'ensemble des termes étendus qu'il permet de dériver contenant exclusivement l'opérateur $\otimes$. Ce langage a pour intérêt d'être décidable par simple comptage des lettres d'un mot, puisque celles-ci peuvent être permutées librement par associativité et commutativité de $\otimes$.

Étant donné une grammaire commutative $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$, le *fragment commutatif* de G est la grammaire commutative $G' = (\mathcal{N}, \Sigma, S, \mathcal{P}')$, dont les productions excluent l'emploi de l'opérateur $\odot$. Ainsi, $\mathcal{P}'$ contient uniquement les productions suivantes :

— Si une production $A(\otimes\ x\ y) \leftarrow B(x)\ C(y)$ appartient à $\mathcal{P}$, alors elle appartient également à $\mathcal{P}'$.
— Si une production $A(a) \leftarrow$ appartient à $\mathcal{P}$, alors elle appartient à $\mathcal{P}'$.

Par suite, le *langage commutatif* d'un non-terminal A selon G est défini comme le langage étendu de A selon le fragment commutatif G' de G. Ce langage étant construit exclusivement à l'aide de l'opérateur $\otimes$, l'ordre des lettres composant ses mots est indifférent : l'image de Parikh d'un mot w de $(\Sigma \cup \mathcal{N})^*$ suffit à décider de son appartenance au langage commutatif de A. Aussi, nous nous intéresserons uniquement dans la suite à l'image de Parikh de ce dernier, notée $\psi(G, A)$; et, par abus de langage, l'expression *langage commutatif de A selon G* désignera directement $\psi(G, A)$.

5.5.2 Algorithme d'analyse pour les CRG

Nous présentons d'abord l'algorithme d'analyse selon une CRG, sous la forme d'un ensemble de règles d'inférence, et détaillons son fonctionnement avant de démontrer ses propriétés. L'algorithme est donné par la figure 5.12 ; il permet de construire des *faits de dérivation* formés comme des quintuplets ou comme le symbole spécial $\top$. Un mot $w = l_1 \dots l_{|w|}$ est reconnu par une CRG $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ si et seulement si le fait $\top$ est dérivable à partir du fait $(S, \vec{0}, S, 0, |w|)$.

$$\frac{(A, \vec{0}, A, i, j) \quad A(\odot\ a\ x) \leftarrow B(x) \in \mathcal{P} \quad a = l_{i+1} \vee a = \varepsilon}{(B, \vec{0}, B, i + |a|, j)}\ \textsc{Analyse}$$

$$\frac{(A, v, B, i, j) \quad B(\otimes\ a\ x) \leftarrow C(x) \in \mathcal{P} \quad \|v\| \leq |w|}{(A, v + \vec{1}_a, C, i, j)}\ \textsc{Hypothèse}$$

$$\frac{(A, v, B, i, j) \quad v + \vec{1}_B \in \psi(G, A) \quad v = \psi(w, i, i') + \psi(w, j', j)}{(B, \vec{0}, B, i', j')}\ \textsc{Confirmation}$$

$$\frac{(A, v, B, i, j) \quad B(a) \leftarrow \in \mathcal{P} \quad v + \vec{1}_a = \psi(w, i, j)}{\top}\ \textsc{Réussite}$$

FIGURE 5.12 – Algorithme d'analyse selon une CRG

L'algorithme procède en déduisant de manière descendante un arbre de dérivation associé au mot w, tout en conservant continuellement en mémoire un fait de dérivation traduisant sa progression dans la reconnaissance de w. Un fait de dérivation est soit le symbole $\top$, soit un quintuplet (A, v, B, i, j), formé d'un symbole non-terminal $A \in \mathcal{N}$, d'un vecteur $v \in \mathbb{N}^{\Sigma \cup \mathcal{N}}$, d'un autre symbole non-terminal $B \in \mathcal{N}$ (possiblement égal à A), et de deux positions $i, j \in [|w|]$ dans le mot w. Le fait $\top$ dénote le succès de l'analyse de w par G, tandis qu'un

quintuplet dénote que le non-terminal A correspond à la position i dans w, et qu'il se réécrit en un non-terminal B après avoir introduit entre i et j, au moyen de l'opérateur $\otimes$, l'ensemble de lettres compté par le vecteur v. Observons que si la dernière étape de la dérivation est formée par une production employant l'opérateur non-commutatif $\odot$, le fait correspondant est toujours de la forme $(A, \vec{0}, A, i, j)$, et que dans ce cas, chaque étape de la dérivation fait progresser i de 1 vers la fin du mot w. La figure 5.13 montre un terme linéaire droit t appartenant au langage $\mathcal{T}(G)$ d'une grammaire G, et son terme équivalent t' dont la frontière est $w = w_1 w'_2 w_3 w'_4 w''_2$. Nous nous appuierons sur cette figure pour détailler le fonctionnement de l'algorithme.

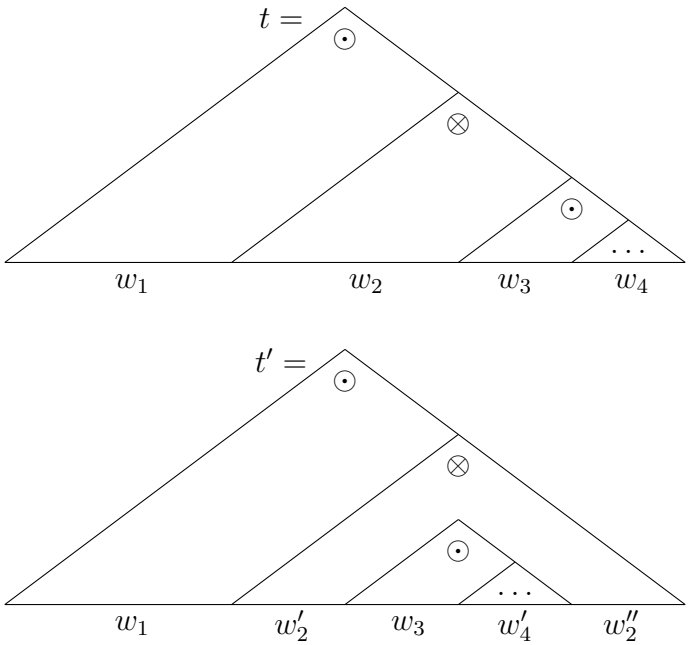


FIGURE 5.13 – Forme générale d'une dérivation CRG

Nous détaillons maintenant chacune des règles de l'algorithme et expliquons son fonctionnement dans la reconnaissance d'un terme.

— La règle ANALYSE correspond à l'emploi d'une production contenant l'opérateur $\odot$. Elle peut s'interpréter comme une transition dans un automate classique : le non-terminal en cours de reconnaissance passe de A à B, et la lettre a est lue dans w, avançant la position courante à $i + |a|$ (i restant constant dans le cas où $a = \varepsilon$). Le vecteur v et la position j, servant à la reconnaissance des productions utilisant $\otimes$, sont inchangés. Sur la figure 5.13, cette règle est employée dans les sous-termes marqués $\odot$ de t, par exemple pour reconnaître le premier facteur w_1 du mot w analysé.

— La règle HYPOTHÈSE devine l'emploi d'une production commutative réécrivant B en $\otimes\ a\ C$. Cette règle permet de descendre dans l'arbre de

dérivation de t, mais ne confirme pas immédiatement la présence de la lettre a dans w : celle-ci peut en effet se trouver à une position arbitraire entre i et j dans w, par ré-association et commutation de $\otimes$ dans t'. Le fait de dérivation résultant accumule cependant une occurrence de la lettre a dans le vecteur v, et met à jour le prochain non-terminal à réécrire de B en C. Observons enfin que l'usage de cette règle est restreint par la condition $\|v\| \leq |w|$, qui limite la taille de v au nombre de lettres dans w (par abus de notation, nous considérons $\vec{1}_\varepsilon = \vec{0}$). Appliquée à la figure 5.13, cette règle permet d'accumuler dans le vecteur v les lettres du facteur w_2.

— La règle Confirmation vérifie le bon emploi d'une série d'occurrences de la règle Hypothèse. Cette règle compte les lettres de w de i à i' et de j' à j, et vérifie que le résultat correspond au vecteur v accumulé par des productions successives. Sur la figure 5.13, l'emploi de cette règle correspond à la transition du terme marqué $\otimes$ à son sous-terme marqué $\odot$: le mot w'_2 est le fragment de w compris entre i et i', et le mot w''_2 est celui allant de j' à j. Ces mots doivent seulement respecter l'invariant $\psi(w'_2) + \psi(w''_2) = \psi(w_2)$; les occurrences de $\otimes$ qui dominent w_2 permettant d'en réorganiser les lettres dans n'importe quel ordre (sans toutefois modifier la structure du sous-terme $\odot$ qui domine w_3).

— La règle Réussite termine la lecture du mot w avec une production terminale, en vérifiant la validité d'un éventuel vecteur v accumulé. Cette règle effectue la même vérification que Confirmation lorsque le dernier opérateur visité est $\otimes$, ou vérifie simplement que la dernière lettre à analyser dans w est bien a dans le cas contraire. Dans tous les cas, elle vérifie l'ensemble des lettres restantes dans w (entre i et j), et termine ainsi l'analyse du mot.

Théorème 5.3. *L'algorithme décrit par la figure 5.12 est correct et complet, et s'exécute de manière non-déterministe en espace logarithmique (*NL*).*

Démonstration. La preuve détaillée de ce résultat s'appuie sur l'explication ci-dessus, en montrant que tout composant d'un fait de dérivation est représentable en espace logarithmique par rapport à $|w|$ (ou au cardinal de $\mathcal{N}$) ; elle figure en annexe dans la section B.2. □

5.5.3 Algorithme d'analyse pour les CCFG

De même que pour les CRG, nous présentons maintenant l'algorithme d'analyse pour les CCFG ; celui-ci est donné par la figure 5.14.

Les faits de dérivation de cet algorithme sont de la forme (v, i, j), se composant d'un vecteur de symboles non-terminaux $v \in \mathbb{N}^{\mathcal{N}}$ et de deux positions dans w : $i, j \in [|w|]$. Un fait (v, i, j) dénote que les symboles non-terminaux

$$\frac{A(l_i) \leftarrow \in \mathcal{P}}{(\vec{1}_A, i-1, i)}\ \textsc{Constante} \qquad \frac{\exists t \in \mathcal{T}_G(A).\, \mathrm{frt}(t) = \varepsilon}{(\vec{1}_A, i, i)}\ \textsc{Vide}$$

$$\frac{(\vec{1}_B, i, j) \quad (\vec{1}_C, j, k) \quad A(\odot\ x\ y) \leftarrow B(x)\ C(y) \in \mathcal{P}}{(\vec{1}_A, i, k)}\ \textsc{Analyse}$$

$$\frac{(u, i, j) \quad (v, j, k) \quad \|u+v\| \leq |w|+1}{(u+v, i, k)}\ \textsc{Combinaison}$$

$$\frac{(v, i, j) \quad v \in \psi(G, A)}{(\vec{1}_A, i, j)}\ \textsc{Réduction}$$

FIGURE 5.14 – Algorithme d'analyse selon une CCFG

contenus dans le vecteur v permettent, dans un certain ordre, de dériver le facteur $l_{i+1} \dots l_j$ de w ; l'algorithme procède de manière ascendante dans l'arbre de dérivation, en construisant des faits successifs décrivant des fragments plus grands de w. Un mot w est reconnu par une CCFG $G = (\mathcal{N}, \Sigma, S, \mathcal{P})$ si et seulement si l'algorithme permet de dériver le fait $(\vec{1}_S, 0, |w|)$. Observons que, suite à un emploi de l'opérateur $\odot$ qui ne permet pas de réordonner ses éléments, le vecteur $v = \vec{1}_A$ contient toujours un unique symbole non-terminal.

Nous détaillons maintenant les différentes règles de l'algorithme :

— La règle CONSTANTE interprète la i^{e} lettre de w selon une production terminale $A(l_i)$. Elle conclut que le fragment de w correspondant est dérivable à partir de A.

— La règle VIDE élimine un sous-terme trivial (ayant pour frontière le mot vide) dérivé à partir de A. Elle permet de raccourcir une dérivation en éliminant en une étape les sous-termes qui ne font pas progresser la reconnaissance de w.

— La règle ANALYSE regroupe deux non-terminaux B et C pouvant se réécrire en deux facteurs contigus de w (de i à j et de j à k respectivement) à l'aide d'une production de $\mathcal{P}$ permettant de réécrire un A comme la concaténation de B et C. Elle en déduit que le facteur de w allant de i à k peut s'interpréter comme un A.

— La règle COMBINAISON assemble deux vecteurs dont le contenu permet de dériver deux facteurs contigus de w, sans tenir compte de l'ordre de leurs éléments. Elle correspond à un emploi de l'opérateur $\otimes$ pour regrouper deux sous-termes également formés avec $\otimes$, et dont les éléments peuvent donc être librement permutés. La borne $\|u+v\| \leq |w|+1$ limite le nombre de symboles non-terminaux utilisés pour former w : chacun d'entre eux doit correspondre à au moins une lettre de w, à l'exception d'au plus un symbole produit par la règle VIDE. Observons que cette

règle ne vérifie pas que G permet de recombiner les éléments de u et de v à l'aide de l'opérateur $\otimes$.

— La règle RÉDUCTION vérifie le bon emploi d'une série d'occurrences de la règle COMBINAISON vis-à-vis de la grammaire G. Elle accepte le contenu du vecteur v comme les éléments (librement recombinables) dérivés à partir d'une occurrence de A, en s'appuyant sur le langage commutatif $\psi(G, A)$ de A selon G, qui est semilinéaire.

Théorème 5.4. *L'algorithme donné par la figure 5.14 est correct et complet, et s'exécute en* LOGCFL.

Démonstration. La preuve de la complexité de cet algorithme s'appuie sur un résultat de Ruzzo [1980] caractérisant la classe de complexité LOGCFL ; son détail est donné en annexe dans la section B.2. □

5.6 Algorithme général

Nous souhaitons maintenant proposer un algorithme de reconnaissance plus général que ceux de la section précédente. L'objectif est de pouvoir traiter les CMCFG et le problème de la reconnaissance universelle, tout en épousant les bornes de difficultés exhibées dans la section 5.4. Nous traiterons également le cas des grammaires dites effaçantes, qui permettent la non-utilisation de certains éléments associés au membre droit d'une production.

Le principal obstacle à la description d'un tel algorithme est la taille minimale d'une dérivation produisant un terme qui reconnaisse un mot w, comme l'illustre la figure 5.15. Une telle grammaire reconnaît simplement le langage $\mathcal{L}(G) = \{\varepsilon\}$, mais dérive des termes de taille exponentielle par rapport au nombre de symboles non-terminaux dans $\mathcal{N}$. Plus généralement, une CMCFG peut dériver des tuples de termes et/ou de contextes « triviaux », c'est à dire qui ne contribuent pas réellement à la formation du mot w à reconnaître. Ce problème ne peut pas être résolu en supprimant simplement les symboles ε de la grammaire G : comme illustré précédemment dans la figure 5.5, la présence de feuilles ε est susceptible d'altérer le langage d'un terme, lorsqu'elles apparaissent sous un opérateur $\odot$ situé entre deux opérateurs $\otimes$.

$$G = (\{A_0 \dots A_n\}, \Sigma, A_0, \mathcal{P})$$

$$\mathcal{P} = \left\{ A_i \left(\begin{array}{c} \otimes \\ / \quad \backslash \\ A_{i+1} \quad A_{i+1} \end{array} \right) \mid 0 \leq i < n \right\} \cup \{A_n(\varepsilon)\}$$

FIGURE 5.15 – Exemple de CCFG dérivant des termes de grande taille

Toutefois cette configuration, qui rompt l'associativité de l'opérateur commutatif $\otimes$, est la seule qui pose problème : l'ajout d'un contexte de la forme $\otimes(\varepsilon, x)$ n'a pas d'impact sur le langage d'un terme, pas plus que l'emploi de plusieurs concaténations successives du symbole ε. Nous proposons par conséquent dans ce chapitre un mécanisme de compression des termes et contextes sur un alphabet commutatif $\mathrm{Comm}(\Sigma)$, qui permet d'éliminer exactement les fragments d'un terme qui n'ont pas d'influence sur son langage associé. Ce mécanisme nous permettra d'analyser un mot selon une grammaire modulo compression du terme intermédiaire, nous permettant d'atteindre exactement les bornes de complexité données dans la section 5.4.

5.6.1 Typage sémantique des termes

Nous proposons un système de typage sémantique pour les lambda-termes représentant des termes et contextes commutatifs. Les types sémantiques atomiques que nous utilisons correspondent à des termes clos, de type syntaxique 0. Nous décrivons les types suivants :

- Le type *epsilon*, $\mathcal{E}$, est associé aux termes dont la frontière se réduit à ε. Ces termes pourront par la suite se voir substituer la constante ε.
- Le type *rigide*, $\mathcal{R}$, est associé aux termes se réduisant à une lettre, ou dont la racine est l'opérateur $\odot$. Intuitivement, ces termes forment un « bloc » solide dont les éléments ne peuvent pas se mélanger avec d'autres issus d'un contexte plus large.
- Le type *commutatif*, $\mathcal{C}$, est associé aux termes dont la racine est l'opérateur $\otimes$. Intuitivement, le langage d'un tel terme dépend de son contexte : des lettres supplémentaires peuvent par exemple s'intercaler entre plusieurs de ses sous-termes par associativité et commutativité de $\otimes$.
- Le type *vide*, $\bot$, est associé aux termes destinés à être ultérieurement supprimés lors de la dérivation. Ce type servira à raccourcir les dérivations dans le cas des grammaires effaçantes.

Familles de types Nous considérons ensuite quatre familles de types linéaires (V_α, E_α, T_α et U_α), permettant de typer des contextes d'arité quelconque. Ces familles sont définies de manière mutuellement récursive comme suit :

$$\begin{array}{lcllcl} V_0 & = & \{\bot\} & V_{\alpha\to\beta} & = & V_\alpha \to V_\beta \\ E_0 & = & \{\mathcal{E}\} & E_{\alpha\to\beta} & = & E_\alpha \to E_\beta \cup V_\alpha \to E_\beta \\ T_0 & = & \{\mathcal{R},\mathcal{C}\} & T_{\alpha\to\beta} & = & U_\alpha \to T_\beta \cup V_\alpha \to T_\beta \\ & & & U_\alpha & = & T_\alpha \cup E_\alpha \end{array}$$

Nous omettons dans la suite le type syntaxique en indice de V, E, T ou U lorsqu'il est quelconque. Les lambda-termes typés par V dénotent des termes ou contextes vides, qui seront effacés durant la dérivation. La famille E correspond à des contextes dont le type résultant est $E_0 = \mathcal{E}$, autrement dit des termes

dont la frontière est ultimement ε. La famille de types auxiliaire T recouvre des lambda-termes qui dénotent ultimement des termes commutatifs dont la frontière n'est pas vide. Enfin, la famille U recouvre l'ensemble des termes ou contextes utiles à une dérivation, c'est à dire les termes de T ou E.

Conventions de notation Par convention, nous utiliserons une police particulière pour les variables $\mathbf{a}, \mathbf{b}, \dots$ qui dénotent des types sémantiques, et emploierons librement dans les prochaines sous-sections le mot « terme » pour désigner des lambda-termes représentant des termes ou des contextes sur une algèbre commutative $\mathrm{Comm}(\Sigma)$. Nous appelons *environnement de typage* une fonction partielle (notée $\Gamma, \Delta, \dots$) associant à un ensemble de lambda-variables $\mathcal{X}$ des types sémantiques de U, et respectant leur type syntaxique en ce sens que $\Gamma(x^\alpha) \in U_\alpha$. Nous abrégeons par simplicité $x \in \mathrm{Dom}(\Gamma)$ en $x \in \Gamma$. Si deux environnements de typage Γ et Δ sont tels que $\mathrm{Dom}(\Gamma) \cap \mathrm{Dom}(\Delta) = \emptyset$, nous écrivons Γ, Δ pour dénoter leur union. Enfin, étant donné une constante c, nous écrirons $\tau(c)$ pour dénoter l'ensemble des types sémantiques de c.

Type des constantes Étant donné une algèbre commutative $\mathrm{Comm}(\Sigma)$ représentée par des lambda-termes, nous donnons aux constantes de l'ensemble $C = \{\odot, \otimes, \varepsilon\} \cup \Sigma$ les types sémantiques suivants :

- La constante ε est de type $\mathcal{E}$.
- Pour tout $a \in \Sigma$, a est de type $\mathcal{R}$.
- L'opérateur $\odot$ est dénoté par une constante pouvant avoir les types suivants : $\mathcal{E} \to \mathcal{E} \to \mathcal{E}$ ou $\mathbf{a} \to \mathbf{b} \to \mathcal{R}$ si $\mathbf{a}$ ou $\mathbf{b}$ est dans T_0.
- L'opérateur $\otimes$ est dénoté par une constante pouvant avoir les types suivants : $\mathbf{a} \to \mathcal{E} \to \mathbf{a}$ ou $\mathcal{E} \to \mathbf{a} \to \mathbf{a}$ si $\mathbf{a} \in U_0$, ou bien $\mathbf{a} \to \mathbf{b} \to \mathcal{C}$ si $\mathbf{a}, \mathbf{b} \in T_0$.
- Pour tout type syntaxique α, la constante supplémentaire Ω^α a le type sémantique V_α.

Nous complétons l'ensemble C des constantes issues de $\mathrm{Comm}(\Sigma)$ par un ensemble de constantes Ω^α représentant un lambda-terme arbitraire destiné à être effacé lors d'une dérivation dans le cadre d'une grammaire effaçante.

Jugements et dérivations de typage Un *jugement de typage* est un triplet $\Gamma \vdash M : \mathbf{a}$ où Γ est un environnement de typage, M est un lambda-terme *affine* (contentant au plus une occurrence de chaque variable) et $\mathbf{a}$ est un type sémantique appartenant à U. Le *sujet* d'un jugement de typage est le lambda-terme M auquel il attribue un type. Un jugement est dit *bien formé* lorsque :

1. Si $x \notin \mathrm{FV}(M)$ et $x \in \Gamma$, alors $\Gamma(x) \in E$.
2. Si $\mathbf{a} \in E$, alors pour tout $x \in \Gamma$, le type $\Gamma(x)$ appartient à E.

La figure 5.16 décrit l'ensemble des règles permettant de dériver des jugements de typage. Nous notons $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ l'existence d'une dérivation

$$\frac{}{\Gamma, x : \mathbf{a} \vdash x : \mathbf{a}}\, var \qquad \frac{c \in C \quad \mathbf{a} \in \tau(c)}{\Gamma \vdash M : \mathbf{a}}\, cst$$

$$\frac{\Gamma, x : \mathbf{a} \vdash M : \mathbf{b}}{\Gamma \vdash \lambda x.M : \mathbf{a} \to \mathbf{b}}\, abs \qquad \frac{\Gamma \vdash M : \mathbf{a} \to \mathbf{b} \quad \Delta \vdash N : \mathbf{a}}{\Gamma \vdash MN : \mathbf{b}}\, app$$

$$\frac{\Gamma \vdash M : \mathbf{b} \quad x \notin \Gamma}{\Gamma \vdash \lambda x.M : V_\alpha \to \mathbf{b}}\, abs, \bot \qquad \frac{\Gamma \vdash M : V_\alpha \to \mathbf{b}}{\Gamma \vdash MN^\alpha : \mathbf{b}}\, app, \bot$$

FIGURE 5.16 – Règles de dérivation pour les types sémantiques

complète $\mathtt{M}$ employant ce système de règles et dont la conclusion est $\Gamma \vdash M : \mathbf{a}$. Le *type* d'une dérivation est, par extension, celui du jugement qui la conclut ; et une dérivation est dite *close* lorsque l'environnement de typage Γ de sa conclusion est vide.

Propriétés Notre système de typage sémantique possède plusieurs propriétés essentielles : renforcement, affaiblissement, réduction et expansion (affine) du sujet. Les preuves détaillées de ces propriétés sont données en annexe (section B.3.1), incluant un lemme de substitution et un lemme d'extraction.

Propriété 5.5 (Renforcement)**.** *Si* $\Gamma, x : \mathbf{a} \vdash M : \mathbf{b}$ *est dérivable et que* x *n'est pas libre dans* M*, alors* $\Gamma \vdash M : \mathbf{b}$ *est dérivable (lemme B.7).*

Propriété 5.6 (Affaiblissement)**.** *Si* $\Gamma \vdash M : \mathbf{b}$ *est dérivable et que le jugement* $\Gamma, \Delta \vdash M : \mathbf{b}$ *est bien formé, alors* $\Gamma, \Delta \vdash M : \mathbf{b}$ *est dérivable (lemme B.8).*

Propriété 5.7 (Réduction du sujet)**.** *Étant donné un terme affine* M *tel que que* $\Gamma \vdash M : \mathbf{b}$ *est dérivable, si* $M \xrightarrow{*}_{\beta\eta} N$ *alors* $\Gamma \vdash N : \mathbf{b}$ *est dérivable (lemme B.10).*

Propriété 5.8 (Expansion affine du sujet)**.** *Étant donné deux termes affines* M *et* N *tels que* $M \xrightarrow{*}_{\beta\eta} N$ *et un environnement de typage* Γ *incluant les variables libres de* M*, s'il existe une dérivation* $\mathtt{N} :: \Gamma \vdash N : \mathbf{a}$*, alors il existe une dérivation* $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ *telle que* $\mathtt{M} \xrightarrow{*}_{\beta\eta} \mathtt{N}$ *(lemme B.12).*

Étant donné une dérivation $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ et un terme N tel que $M \xrightarrow{*}_{\beta\eta} N$, si $\mathtt{N} :: \Gamma \vdash N : \mathbf{a}$ est la dérivation obtenue par réduction du sujet M, nous écrirons directement dans la suite que $\mathtt{M} \xrightarrow{*}_{\beta\eta} \mathtt{N}$.

5.6.2 Le système de réécriture $\to_\varepsilon$

Nous introduisons maintenant une relation de réécriture $\to_\varepsilon$ sur les dérivations de types sémantiques, dont les règles sont données par la figure 5.17 ;

nous désignerons ces règles en les numérotant de 1 à 8 (de haut en bas) dans la suite du texte. Ce système de réécriture permet de compresser les termes et contextes apparaissant dans les dérivations d'une CMCFG sans altérer le langage du terme final : les opérations introduisant des occurrences inutiles de ε sont éliminées (par les paires de règles 3,4 et 6,7), et les termes et contextes dont le système de typage garantit la réduction à ε sont réduits à leur plus simple expression (par les règles 1,5 et 8) ; enfin, les termes et contextes typés par V, destinés à être effacés par β-réduction, sont remplacés par une constante Ω de taille fixe (règle 2). Observons que la relation $\to_\varepsilon$ ne modifie que le sujet d'un jugement de typage : son environnement de typage et le type sémantique qui lui sont attribués demeurent constants.

Propriétés de $\to_\varepsilon$ et $\to_{\beta\varepsilon}$ Le système de réécriture $\to_\varepsilon$ est confluent et fortement normalisant, induisant l'existence d'une forme normale unique. En outre, le sujet M de la conclusion d'une dérivation en forme ε-normale est affine, et toutes ses variables libres appartiennent au domaine de son environnement de typage Γ ; ces deux dernières propriétés servent uniquement à démontrer d'autres résultats intermédiaires, et sont établies en annexe par les lemmes B.15 et B.14.

Propriété 5.9 (Forme ε-normale)**.** *Le système de réécriture $\to_\varepsilon$ induit pour toute dérivation une unique forme normale (lemme B.13).*

Si $\mathtt{D} :: \Gamma \vdash M : \mathbf{a}$ est une dérivation sémantique de sujet M, sa forme ε-normale est notée $|\mathtt{D}|_\varepsilon$; par extension le sujet de cette dernière dérivation est dit en forme ε-normale et noté $|M|_\varepsilon$.

Nous nous intéressons ensuite à l'interaction entre ε et β-réduction. En utilisant l'extension de la notion de β-réduction aux dérivations de typage (conséquence de la propriété de réduction du sujet), nous pouvons montrer la normalisation forte et la confluence du système joint $\to_{\beta\varepsilon}$. Une autre propriété de ce système, cruciale dans le fonctionnement de notre algorithme d'analyse modulo compression, est la possibilité de réordonner deux réductions successives $\to_\beta$ et $\to_\varepsilon$: aucun ε-redex n'est créé par β-contraction, entraînant que la β-réduction préserve l'ε-normalité (en d'autres termes, un lambda-terme déjà compressé demeure compact lors de son évaluation par $\to_\beta$).

Propriété 5.10 (Réordonnement de $\to_\beta/\to_\varepsilon$)**.** *Considérons les deux termes affines M et N et leurs dérivations associées $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ et $\mathtt{N} :: \Gamma \vdash N : \mathbf{a}$, tels que $\mathtt{M} \to_\beta \mathtt{N}$ selon la procédure du lemme B.10. Si $\mathtt{N} \to_\varepsilon \mathtt{N}' :: \Gamma \vdash N' : \mathbf{a}$, alors il existe un terme M' muni d'une dérivation $\mathtt{M}' :: \Gamma \vdash M' : \mathbf{a}$ de sorte que $\mathtt{M} \to_\varepsilon \mathtt{M}'$ et $\mathtt{M}' \overset{*}{\to}_\beta \mathtt{N}'$ (lemme B.16). Cette relation est illustrée par le diagramme*

$$\frac{\mathtt{M}\,::}{\Gamma \vdash M : \mathcal{E}} \longrightarrow_{\varepsilon} \frac{}{\Gamma \vdash \varepsilon : \mathcal{E}}\, cst \qquad (M \neq \varepsilon) \tag{1}$$

$$\frac{\dfrac{\mathtt{M}\,::}{\Gamma \vdash M : V_\alpha \to \mathbf{b}}}{\Gamma \vdash MN^\alpha : \mathbf{b}}\, app, \bot \longrightarrow_{\varepsilon} \frac{\dfrac{\mathtt{M}\,::}{\Gamma \vdash M : V_\alpha \to \mathbf{b}}}{\Gamma \vdash M\Omega^\alpha : \mathbf{b}}\, app, \bot \qquad (N^\alpha \neq \Omega^\alpha) \tag{2}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{R} \to \mathcal{E} \to \mathcal{R}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{R}, y : \mathcal{E} \vdash x : \mathcal{R}}\, var}{\Gamma, x : \mathcal{R} \vdash \lambda y.x : \mathcal{E} \to \mathcal{R}}\, abs}{\Gamma \vdash \lambda xy.x : \mathcal{R} \to \mathcal{E} \to \mathcal{R}}\, abs \tag{3}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{E} \to \mathcal{R} \to \mathcal{R}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{R} \vdash y : \mathcal{R}}\, var}{\Gamma, x : \mathcal{E} \vdash \lambda y.y : \mathcal{R} \to \mathcal{R}}\, abs}{\Gamma \vdash \lambda xy.y : \mathcal{E} \to \mathcal{R} \to \mathcal{R}}\, abs \tag{4}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{E} \vdash \varepsilon : \mathcal{E}}\, cst}{\Gamma, x : \mathcal{E} \vdash \lambda y.\varepsilon : \mathcal{E} \to \mathcal{E}}\, abs}{\Gamma \vdash \lambda xy.\varepsilon : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, abs \tag{5}$$

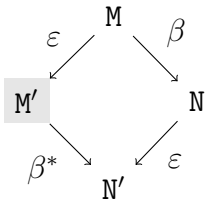
$$\frac{}{\Gamma \vdash \otimes : \mathbf{a} \to \mathcal{E} \to \mathbf{a}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathbf{a}, y : \mathcal{E} \vdash x : \mathbf{a}}\, var}{\Gamma, x : \mathbf{a} \vdash \lambda y.x : \mathcal{E} \to \mathbf{a}}\, abs}{\Gamma \vdash \lambda xy.x : \mathbf{a} \to \mathcal{E} \to \mathbf{a}}\, abs \qquad (\mathbf{a} \neq \mathcal{E}) \tag{6}$$

$$\frac{}{\Gamma \vdash \otimes : \mathcal{E} \to \mathbf{a} \to \mathbf{a}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathbf{a} \vdash y : \mathbf{a}}\, var}{\Gamma, x : \mathcal{E} \vdash \lambda y.y : \mathbf{a} \to \mathbf{a}}\, abs}{\Gamma \vdash \lambda xy.y : \mathcal{E} \to \mathbf{a} \to \mathbf{a}}\, abs \qquad (\mathbf{a} \neq \mathcal{E}) \tag{7}$$

$$\frac{}{\Gamma \vdash \otimes : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, cst \longrightarrow_{\varepsilon} \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{E} \vdash \varepsilon : \mathcal{E}}\, cst}{\Gamma, x : \mathcal{E} \vdash \lambda y.\varepsilon : \mathcal{E} \to \mathcal{E}}\, abs}{\Gamma \vdash \lambda xy.\varepsilon : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, abs \tag{8}$$

FIGURE 5.17 – Le système de réécriture $\to_\varepsilon$

en diamant suivant :



Propriété 5.11 (Forme $\beta\varepsilon$-normale)**.** *Le système $\to_{\beta\varepsilon}$ est confluent et induit une forme normale unique (lemme B.19).*

Comme pour le système $\to_\varepsilon$, nous notons respectivement $|\mathtt{D}|_{\beta\varepsilon}$ et $|M|_{\beta\varepsilon}$ la forme $\beta\varepsilon$-normale d'une dérivation $\mathtt{D}$ et d'un terme M possédant un type sémantique.

Contraintes sur la taille des termes Nous établissons maintenant plusieurs contraintes sur la taille des termes en forme ε-normale, dans le but de montrer ultérieurement qu'un terme (ou contexte) commutatif peut, modulo ε-conversion, être représenté en espace polynomial relativement à la longueur des mots de son langage. Ces résultats nous serviront à établir la complexité de l'algorithme d'analyse général à l'issue de ce chapitre.

Nous définissons d'abord plusieurs mesures quantifiant la taille d'un terme et celle de son type syntaxique :

Définition 5.8. La *taille* $|M|$ d'un lambda-terme M est définie inductivement comme suit :
- $|x| = 1$
- $|c| = 1$ pour toute constante $c \in C$
- $|\lambda x.P| = |P|$
- $|MN| = |M| + |N|$

La *taille concrète* $\|M\|$ d'un lambda-terme M représentant un terme commutatif est définie inductivement comme suit :
- $\|x\| = 0$
- $\|c\| = \begin{cases} 3 \text{ si } c \in \{\odot, \otimes\} \\ 1 \text{ si } c \in \Sigma \\ 0 \text{ si } c \in \{\varepsilon, \Omega^\alpha\} \end{cases}$
- $\|\lambda x.P\| = \|P\|$
- $\|MN\| = \|M\| + \|N\|$

La *taille* $|\tau|$ d'un type syntaxique τ est définie inductivement comme suit :
- $|0| = 1$
- $|\alpha \to \beta| = |\alpha| + |\beta|$

Tout lambda-terme M peut aisément être représenté en espace polynomial par rapport à sa taille $|M|$, et sa taille concrète $\|M\|$, lorsqu'il est compressé, est proportionnelle à la longueur des mots qu'il reconnaît.

Nous établissons maintenant une propriété notable des termes en forme $\beta\varepsilon$-normale, qui est que leur taille est bornée linéairement par leur taille concrète et par la taille de leur type : ainsi, l'espace requis pour représenter un terme compressé est borné par longueur des mots de son langage. Une autre propriété utile à notre algorithme de reconnaissance général est que la substitution et la β-réduction de termes ε-normaux préserve leur taille concrète. Cette dernière propriété entrera en jeu lors de l'application d'une production : la somme des tailles des termes compressés associés aux non-terminaux résultants sera toujours inférieure ou égale à celle des termes associés au non-terminal de départ.

Propriété 5.12. *Si* $\mathtt{M}$ *est une dérivation en forme* $\beta\varepsilon$*-normale typant un terme clos* M *de type* α*, alors* $|M| \leq \|M\| + |\alpha|$ *(corollaire B.24).*

Propriété 5.13. *Soit* $M, N, N_1, \ldots, N_n$ *des termes respectant ces conditions :*
- $\mathrm{FV}(M) = \{x_1 \ldots x_n\}$
- $M[N_i/x_i] \xrightarrow{*}_\beta N$
- *Il existe des dérivation* ε*-normales bien formées des jugements suivants :*

$$\Gamma, x_1 : \mathbf{a_1} \ldots x_n : \mathbf{a_n} \vdash M : \mathbf{b} \qquad \vdash N : \mathbf{b} \qquad \vdash N_1 : \mathbf{a_1} \;\ldots\; \vdash N_n : \mathbf{a_n}$$

L'égalité suivante est respectée (lemme B.25) :

$$\sum_{i=1}^{n} \|N_i\| = \|N\| - \|M\|$$

5.6.3 Algorithmes d'analyse NP

Le système de réécriture $\to_\varepsilon$ et ses propriétés induisent une relation d'équivalence sur nos termes commutatifs : celle-ci garantit pour tout terme l'existence d'une forme normale de taille polynomiale par rapport à la longueur des mots de son langage, et supprime les sous-termes dont l'élimination par β-réduction est assurée. De plus, le langage d'un terme t est le même que celui de sa forme normale $|t|_\varepsilon$ (par abus de notation, nous étendons la relation $\to_\varepsilon$ sur les lambda-termes aux termes commutatifs qu'ils dénotent). Par suite, étant donné une grammaire commutative G et un mot w, des procédures construisant la forme ε-normale $|t|_\varepsilon$ d'un terme t tel que $t \in \mathcal{T}(G)$ et $w \in \mathcal{L}(t)$ permettent d'établir les résultats de complexité attendus.

Analyse universelle des CCFG L'analyse universelle d'un mot selon une CCFG (et, par conséquent, selon une CRG) peut s'effectuer en temps polynomial sur une machine non-déterministe. Soit $G = (\Sigma, \mathcal{N}, S, \mathcal{P})$ une CCFG, que

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP	
CCFG	LogCFL-C	NP	
CMG	NP	PSPACE-D	EXPTIME
CMREG	NP	PSPACE-C	EXPTIME-C
CMCFG	NP	PSPACE-D	EXPTIME-C

Table 5.6 – Résultats de complexité de la sous-section 5.6.3

l'on suppose en forme normale (voir page 133), et w un mot de Σ^* à analyser selon G, nous considérons le raisonnement suivant :

1. Tout non-terminal de G dans une dérivation est associé à un unique terme, dont le type syntaxique est 0. Le système de typage sémantique décrit précédemment permet alors de raffiner les types rattachés aux non-terminaux, et de produire une grammaire G', produisant le même langage de termes $\mathcal{T}(G') = \mathcal{T}(G)$, mais dans laquelle le type sémantique des termes produits par chaque symbole non-terminal (à l'exception de l'axiome) est connu. Plus précisément, nous construisons $G' = (\Sigma, \mathcal{N}', \sigma, \mathcal{P}')$ ainsi :
   — L'axiome σ est un symbole non-terminal nouvellement introduit.
   — L'ensemble $\mathcal{N}'$ des non-terminaux de G' contient σ, et est formé pour le reste en associant aux symboles de $\mathcal{N}$ un type sémantique : $\mathcal{N}' = (\mathcal{N} \times \{\mathcal{R}, \mathcal{C}, \mathcal{E}\}) \cup \{\sigma\}$. Nous noterons le type sémantique de ces symboles en exposant (*e.g.* $A^{\mathcal{R}}$ lorsque $A \in \mathcal{N}$).
   — Les productions de $\mathcal{P}'$ étendent les productions de $\mathcal{P}$ en respectant le typage sémantique introduit par les non-terminaux de $\mathcal{N}'$. Ainsi, si $A(\odot\ x\ y) \leftarrow B(x)\ C(y)$ est une production de $\mathcal{P}$, alors $\mathcal{P}'$ contient toutes les productions de la forme $A^{\alpha}(\odot\ x\ y) \leftarrow B^{\beta}(x)\ C^{\gamma}(y)$ pour lesquelles $\beta \to \gamma \to \alpha$ est un type sémantique possible de la constante $\odot$ (voir page 140). De même, les productions terminales de $\mathcal{P}'$ sont de la forme $A^{\mathcal{E}}(\varepsilon) \leftarrow$ ou $A^{\mathcal{R}}(a) \leftarrow$. Finalement, $\mathcal{P}'$ contient trois productions de la forme $\sigma(x) \leftarrow S^{\alpha}(x)$, avec $\alpha \in \{\mathcal{R}, \mathcal{C}, \mathcal{E}\}$.
2. En appliquant les règles du système de réécriture $\to_{\varepsilon}$ aux productions de G', nous obtenons une grammaire G'' dont le langage de termes contient les formes ε-normales des termes de $\mathcal{T}(G)$. Les non-terminaux correspondant à des termes de type $\bot$ (ceux dont les variables sont absentes du membre gauche après ε-réduction) sont exclus du membre droit de la production. Aussi, la grammaire résultante n'est pas généralement en forme normale : ainsi, une production $A^{\mathcal{R}}(\odot\ x\ y) \leftarrow B^{\mathcal{R}}(x)\ C^{\mathcal{E}}(y)$ présente dans G' est remplacée par $A^{\mathcal{R}}(x) \leftarrow B^{\mathcal{R}}(x)$ dans G''. De plus,

puisque le système de réécriture $\to_\varepsilon$ préserve le langage des termes commutatifs qu'il réécrit, et que tout terme reconnu par G'' correspond, par ε-réduction, à un terme de G, les langages de mots de ces deux grammaires sont identiques.

3. Si $w \in \mathcal{L}(G)$, il existe donc un terme $t \in \mathcal{T}(G'')$ tel que $w \in \mathcal{L}(t)$. Construire et analyser un tel terme selon G'' permet donc de décider si $w \in \mathcal{L}(G)$.

Les étapes 1 et 2, consistant à construire successivement G' et G'', peuvent être effectuées en temps linéaire par rapport à la taille de G. La grammaire G'' peut, au besoin, être mise en forme normale (notamment en supprimant les règles unitaires) en temps polynomial. Le terme t de l'étape 3 étant ε-normal, sa taille est polynomiale en $|w|$ par la propriété 5.12, et il peut donc être construit en temps non-déterministe polynomial. Par suite, la reconnaissance de t par G'' revient à l'analyse d'un terme par une grammaire régulière de termes, laquelle s'effectue en temps polynomial. Par conséquent le problème de l'analyse universelle selon une CCFG (et, par extension, selon une CRG) est décidable et appartient à NP.

Analyse fixée des CMCFG Nous considérons maintenant la complexité de l'analyse pour une CMCFG fixée. Comme dans le cas précédent, celle-ci est obtenue en calculant une grammaire équivalente à G reconnaissant les formes ε-normales des termes de $\mathcal{T}(G)$, puis en devinant un terme $|t|_\varepsilon$ dont la taille est polynomiale par rapport à $|w|$. À la différence des CCFG, les types des non-terminaux d'une CMCFG ne permettent pas de garantir que la taille de la grammaire résultante est polynomiale par rapport à G : cette procédure n'est donc applicable que si la taille de G est fixée à l'avance. L'analyse universelle, abordée dans la prochaine sous-section, requiert de compresser dynamiquement le terme de $\mathcal{T}(G)$ qui reconnaît w.

Comme pour les CCFG, nous construisons une grammaire G' où les non-terminaux sont étiquetés par des types (ou des tuples de types) sémantiques ; observons que le nombre de types sémantiques possibles pour un non-terminal donné est exponentiel dans la taille de son type syntaxique. Nous normalisons ensuite cette grammaire de la même manière que les CCFG précédemment. La taille de la grammaire étant fixée, ces opérations s'effectuent en temps constant par rapport à $|w|$. Il suffit ensuite de deviner un terme ε-normal t (de taille polynomiale en $|w|$), de l'analyser selon la grammaire nouvellement calculée et de vérifier que $w \in \mathcal{L}(t)$ (décidable en NP) pour décider si $w \in \mathcal{L}(G)$ en NP.

5.6.4 Algorithme d'analyse général

En nous appuyant sur le système de réécriture $\to_\varepsilon$ permettant de compresser des termes commutatifs, ainsi que sur les propriétés établies dans cette

section, nous présentons un algorithme général permettant de déterminer si un tuple de lambda-termes (représentant des termes ou des contextes commutatifs) appartient, modulo ε-compression, au langage d'une grammaire commutative G. Décider si un mot w appartient au langage de G revient alors à deviner un terme $|t|_\varepsilon$ compressé (dont la taille est proportionnelle à $|w|$) tel que $w \in \mathcal{L}(t)$, et à déterminer si t appartient, modulo ε-compression, à $\mathcal{T}(G)$ (la construction de $|t|_\varepsilon$ pouvant se faire en NP).

L'algorithme que nous décrivons traite également le cas des grammaires commutatives effaçantes (mais pas celui des grammaires parallèles) ; son fonctionnement s'inspire de l'algorithme de reconnaissance proposé par Kaji *et al.* [1992] pour les MCFG avec ou sans effacement. Il s'appuie sur une notion de buts, formés par un symbole non-terminal associé à un tuple de termes en forme $\beta\varepsilon$-normale, qui doivent être établis comme solvables pour que l'analyse réussisse. L'algorithme procède en deux phases : premièrement, il calcule en EXPTIME (dans le pire cas) un ensemble de buts triviaux pouvant être éliminés par la grammaire considérée, indépendamment du mot à reconnaître ; deuxièmement, il tente de résoudre en PSPACE un ensemble de buts liés au mot à reconnaître, en les décomposant successivement en sous-buts de taille plus réduite. Durant la seconde phase, l'algorithme peut éliminer immédiatement tout but trivial calculé durant la première phase.

Classe	Analyse fixée	Analyse universelle	
		Non-effaçante	Effaçante
Terme	$\mathcal{O}(1)$	NP-C	
CRG	NL	NP-C	
CCFG	LogCFL-C	NP-C	
CMG	NP-C	PSPACE-D	EXPTIME
CMREG	NP-C	PSPACE	EXPTIME
CMCFG	NP-C	PSPACE-D	EXPTIME

Table 5.7 – Résultats de complexité de la sous-section 5.6.4

Nous introduisons les notations et définitions suivantes : étant donné une dérivation $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$, nous écrivons $\mathcal{L}(\mathtt{M})$ pour l'ensemble des λ-termes M' tels qu'il existe une dérivation $\mathtt{M}' :: \Gamma \vdash M' : \mathbf{a}$ et que $\mathtt{M}' \xrightarrow{*}_{\beta\varepsilon} \mathtt{M}$. En outre :

Définition 5.9. Un *élément* $\mathtt{E}$ d'un but est défini comme étant soit une dérivation $\beta\varepsilon$-normale d'un terme clos, soit un symbole spécial $\bot^\alpha$, où α est un type syntaxique. Dans ce dernier cas, l'élément $\mathtt{E}$ est dit *indéfini*.

Le *type syntaxique* d'un élément est, suivant le cas, soit le type syntaxique du sujet de sa dérivation, soit α ; et par abus de langage, lorsqu'un élément $\mathtt{E}$ est défini, son *type* (sémantique) sera le type $\mathbf{a}$ que cette dernière associe à son sujet, et nous noterons de façon transparente $\mathtt{E} :: \Gamma \vdash M : \mathbf{a}$.

Si A est un symbole non-terminal de type $[\alpha_1, \ldots, \alpha_n]$, alors $A(\mathtt{M_1}, \ldots, \mathtt{M_n})$ est un *but*, dans lequel chaque $\mathtt{M_i}$ est un élément de type syntaxique α_i.

Un but $A(\mathtt{M_1}, \ldots, \mathtt{M_n})$ est dit *solvable* lorsqu'il existe un tuple $(P_1, \ldots, P_n)$ appartenant à $\mathcal{L}(A)$ tel que chaque $\mathtt{M_i}$ est soit $\bot^{\alpha_i}$, soit une dérivation telle que $P_i \in \mathcal{L}(\mathtt{M_i})$. En d'autres termes, les buts solvables représentent, là où ils sont définis, des tuples de termes qui appartiennent au langage de leur symbole non-terminal modulo ε-compression.

Enfin, une dérivation dont le sujet est M est dite *triviale* lorsque $\|M\| = 0$ (M est sans constantes hormis ε), et un but $A(\mathtt{M_1}, \ldots, \mathtt{M_n})$ est dit *trivial* lorsque chaque $\mathtt{M_i}$ est soit $\bot^{\alpha_i}$, soit une dérivation triviale.

Première phase Puisque la taille d'une dérivation dépend linéairement de celle de son sujet, la propriété 5.12 implique que l'ensemble des buts triviaux solvables par une grammaire est de taille exponentielle par rapport à celle-ci (les buts triviaux étant formés par des dérivations dont le sujet a une taille concrète nulle, et dont la taille du type syntaxique est fixée par la grammaire). Nous présentons donc un algorithme de point fixe calculant en EXPTIME l'ensemble des buts triviaux solvables d'une grammaire. Une fois construit, l'appartenance d'un but à cet ensemble peut être testée en temps logarithmique par rapport à la taille de ce dernier, soit en PTIME dans notre cas.

Le pré-calcul des buts triviaux est nécessaire pour garantir que la seconde phase de l'algorithme s'effectue en espace polynomial. La complexité algorithmique de cette première phase peut en outre être considérée indépendamment de celle de la seconde, ce qui nous permettra notamment d'établir que l'analyse universelle d'une CMREG non-effaçante s'effectue en PSPACE (le nombre d'éléments triviaux possibles dont le type syntaxique est de la forme $[0, \ldots, 0]$ étant, pour sa part, polynomial).

Soit $G = (\Sigma, \mathcal{N}, S, \mathcal{P})$ une grammaire commutative ; toute règle r de $\mathcal{P}$ est alors de la forme :

$$r = A(M_1^{\alpha_1}, \ldots, M_n^{\alpha_n}) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}) \ldots B_p(x_{p,1}, \ldots, x_{p,n_p})$$

où le non-terminal A a le type $[\alpha_1, \ldots, \alpha_n]$, et chaque non-terminal B_j pour $j \in [p]$ a le type $[\beta_{j,1}, \ldots, \beta_{j,n_j}]$. Étant donné un ensemble T de buts triviaux (initialement vide), notre algorithme calcule l'ensemble $\mathscr{T}(T, G)$ des buts triviaux que G dérive à partir de T. Cet ensemble est formé par l'union des ensembles $\mathscr{T}(T, r)$, décrivant l'ensemble des buts triviaux dérivables à partir de T pour chaque règle $r \in \mathcal{P}$ de G. L'algorithme que nous décrivons calcule simplement un point fixe en faisant croître l'ensemble T des buts triviaux dérivables par la grammaire G : nous posons $\mathscr{T}_0 = \emptyset$ et $\mathscr{T}_{n+1} = \mathscr{T}_n \cup \mathscr{T}(\mathscr{T}_n, G)$; la séquence $(\mathscr{T}_k)_{k \in \mathbb{N}}$ est stationnaire à partir d'un certain k (puisque l'ensemble des buts triviaux solvables par G est fini), et nous notons $\mathscr{T}(G)$ le point fixe ainsi obtenu.

Considérons une règle r de $\mathcal{P}$ ayant la forme décrite plus haut ; suivant la convention décrite page 119 nous employons dans la suite les variables i, j et k pour itérer sur $[n]$, $[p]$ et $[n_j]$ respectivement. Étant donné un ensemble de buts triviaux T, l'ensemble $\mathscr{T}(T, r)$ des buts triviaux dérivables à partir de T selon r est défini comme suit. Un but trivial $A(\mathtt{D_1}, \ldots, \mathtt{D_n})$ appartient à l'ensemble $\mathscr{T}(T, r)$ si et seulement T contient p buts triviaux $B_j(\mathtt{N'_{j,1}}, \ldots, \mathtt{N'_{j,n_j}})$ et qu'il existe n environnements de typage Γ_i vérifiant les conditions suivantes pour tout j, k :

1. $\Gamma_i(x_{j,k}) = \boldsymbol{b_{j,k}}$ implique que l'élément $\mathtt{N'_{j,k}}$ est une dérivation, conclue par $\mathtt{N'_{j,k}} :: \vdash N'_{j,k} : \boldsymbol{b_{j,k}}$ (cohérence des types)
2. $\mathtt{N'_{j,k}} = \bot^{\beta_{j,k}}$ implique que $x_{j,k}$ n'appartient au domaine d'aucun des Γ_i (non-utilisation des éléments indéfinis)

et que, finalement, les n éléments $\mathtt{D_i}$ de ce but sont tels que :

3. $\mathtt{D_i} = \bot^{\alpha_i}$ implique que le domaine de Γ_i est vide (non-effacement des éléments définis)
4. $\mathtt{D_i} :: \vdash Q_i : \boldsymbol{a_i}$ implique qu'il existe une dérivation $\mathtt{M_i} :: \Gamma_i \vdash M_i : \boldsymbol{a_i}$ telle que, pour tout j, k tel que $\Gamma_i(x_{j,k})$ est défini, $\mathtt{M_i}\left[x_{j,k} \leftarrow \mathtt{N'_{j,k}}\right] \xrightarrow{*}_{\beta\varepsilon} \mathtt{D_i}$ (application de la règle r pour construire les éléments du prochain but)

Notons que la substitution d'une variable par une dérivation dans une dérivation, apparaissant dans la condition 4, correspond à un emploi du lemme de substitution (voir B.9) pour obtenir la dérivation résultante.

Le calcul de $\mathscr{T}(G)$ s'effectue en appliquant itérativement la définition précédente, suivant la procédure décrite plus haut, jusqu'à obtenir un ensemble stable de buts triviaux. Cet ensemble est exactement l'ensemble des buts triviaux solvables par la grammaire G, la preuve de cette affirmation figurant en annexe, dans le théorème B.26. En outre, le calcul de $\mathscr{T}(G)$ s'effectue en temps exponentiel dans le cas général, et polynomial dans le cas d'une CMREG non-effaçante, ce que montre le théorème B.28.

Seconde phase Nous présentons maintenant un algorithme descendant qui, étant donné un ensemble de buts, détermine en PSPACE si cet ensemble peut être effacé en appliquant itérativement certaines règles de réécritures sur son contenu : l'algorithme choisit un but à chaque étape, et peut soit le décomposer en un ensemble (potentiellement vide) de sous-buts en appliquant une règle de $\mathcal{P}$, soit l'éliminer immédiatement s'il s'agit d'un but trivial solvable de $\mathscr{T}(G)$, calculé durant la première phase.

Un ensemble de buts S est qualifié d'*effaçable* s'il peut être réécrit en l'ensemble vide par une série d'applications successives des deux règles suivantes :

— Réécrire S en $S \setminus \{A(\mathtt{D_1} \ldots \mathtt{D_n})\}$, si $\{A(\mathtt{D_1} \ldots \mathtt{D_n})\}$ est un but trivial solvable appartenant à $\mathscr{T}(G)$.

— Réécrire S en $S \setminus \{A(\mathtt{D_1} \dots \mathtt{D_n})\} \cup \Big\{ B_1(\mathtt{N'_{1,1}} \dots \mathtt{N'_{1,n_1}}) \dots B_p(\mathtt{N'_{p,1}} \dots \mathtt{N'_{p,n_p}}) \Big\}$, si il existe n environnements de typage $\Gamma_1 \dots \Gamma_n$ et une règle $r \in \mathcal{P}$ de la forme :

$$r = A(M_1^{\alpha_1}, \dots, M_n^{\alpha_n}) \leftarrow B_1(x_{1,1}, \dots, x_{1,n_1}) \dots B_p(x_{p,1}, \dots, x_{p,n_p})$$

tels que les quatre conditions suivantes (identiques à celles de la première phase) sont respectées :

1. si $\Gamma_i(x_{j,k}) = \boldsymbol{b_{j,k}}$, alors $\mathtt{N'_{j,k}} :: \vdash N'_{j,k} : \boldsymbol{b_{j,k}}$ (cohérence des types)
2. si $\mathtt{N'_{j,k}} = \bot^{\beta_{j,k}}$, alors $x_{j,k} \notin \mathrm{Dom}(\Gamma_i)$ (abandon des éléments $\bot$)
3. si $\mathtt{D_i} = \bot^{\alpha_i}$, alors $\mathrm{Dom}(\Gamma_i) = \emptyset$ (préservation des éléments non-$\bot$)
4. si $\mathtt{D_i} :: \vdash Q_i : \boldsymbol{a_i}$, alors il existe une dérivation $\mathtt{M_i} :: \Gamma_i \vdash M_i : \boldsymbol{a_i}$ et la substitution simultanée de tous les $x_{j,k}$ appartenant au domaine de Γ_i donne $\mathtt{S_i} = \mathtt{M_i}\left[x_{j,k} \leftarrow \mathtt{N'_{j,k}}\right] \overset{*}{\rightarrow}_{\beta\varepsilon} \mathtt{D_i}$ (application de r)

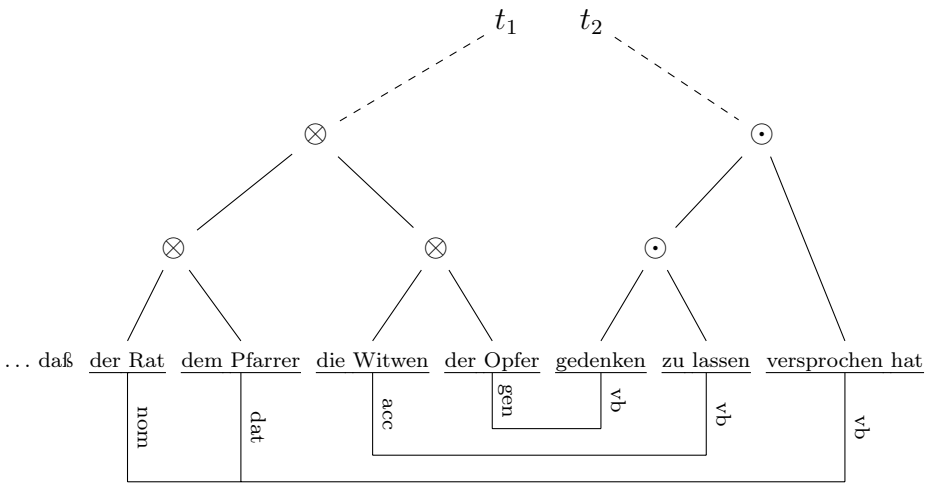
Un but individuel $A(\mathtt{D_1}, \dots, \mathtt{D_n})$ est dit effaçable lorsque le singleton qu'il forme est effaçable. Nous établissons maintenant qu'un but est solvable si et seulement si il est effaçable, et qu'il est possible de décider si un but est effaçable avec une complexité en espace polynomiale par rapport à sa taille : la preuve de ces affirmations est donnée par les théorèmes B.27 et B.29.

5.7 Conclusion

En résumé, nous avons proposé dans ce chapitre un mécanisme algébrique permettant de représenter de manière compacte des phrases où l'ordre des mots est partiellement ou totalement libre. Cet outil a donné lieu à une hiérarchie de grammaires dites commutatives, pour lesquelles nous avons étudié en profondeur la complexité de l'analyse, et dégagé des classes algorithmiquement abordables. Nous avons également proposé et prouvé des algorithmes pour l'analyse de ces différentes classes. Combinée au formalisme décrit dans le chapitre 3, cette représentation peut permettre à une linéarisation de décrire plus exactement, de façon compacte, l'ensemble des énoncés acceptables dans une langue incluant des phénomènes d'ordre libre. Ce travail reste cependant à l'heure actuelle une tâche à accomplir.

En dépit de cela, certains résultats de nature linguistique liés à cette représentation sont d'ores et déjà apparents. En particulier, la modélisation simple d'une langue permettant l'ordonnancement libre de ses syntagmes (telle que le latin) ne semble pas poser de problème majeur, même si la modélisation de phénomènes plus complexes telles que les disjonctions (l'inclusion d'un mot dans une proposition étrangère pour des raisons stylistiques, *cf.* Marouzeau [1922]) peut s'avérer plus délicate.

En revanche, le phénomène de *scrambling* en allemand, qui a originellement motivé le travail présenté dans ce chapitre, soulève un problème intéressant vis-à-vis de la complexité algorithmique de son analyse : le lien entre sa structure profonde et sa réalisation ne semble pas pouvoir être représenté par une CCFG, pour les mêmes raisons que les dépendances croisées du néerlandais échappent au pouvoir de génération forte d'une CFG ; il est en revanche parfaitement capturé par l'emploi d'une CMG employant des contextes unaires, ou d'une CMREG construisant des paires de termes. Rappelons que la complexité de l'analyse à grammaire fixée pour ces dernières est NP-difficile, alors celle des CCFG est identique à celle des grammaires hors-contexte.



(der Opfer, gedenken) → (⊗ die Witwen der Opfer, ⊙ gedenken zu lassen) →
(⊗ (⊗ der Rat dem Pfarrer) (⊗ die Witwen der Opfer), ⊙ (⊙ gedenken zu lassen) versprochen hat)

FIGURE 5.18 – Structure commutative et *scrambling*

Cet argument est illustré par la figure 5.18, reprenant l'un des exemples donné au début de ce chapitre : les termes dessinés au dessus de l'énoncé représentent sa structure phonologique, où la chaîne de verbes présente à la fin de la phrase forme un sous-terme séquentiel t_2, tandis que l'ensemble de leurs arguments est groupé dans un contexte purement commutatif pour former t_1 ; la structure abstraite de l'énoncé figure pour sa part en dessous de celui-ci, formant une série d'arbres enchâssés. Plus bas se trouve une série de réalisations phonologiques possibles pour une telle phrase dans le cadre d'une CMREG : chaque proposition successive est réalisée comme une paire de termes commutatifs, dont le premier contient les arguments verbaux, et le second les verbes eux-mêmes. Ce procédé de linéarisation est remarquablement similaire à celui utilisé dans le chapitre 4 pour rendre compte de l'ordre des mots en néerlandais.

Conclusion

La contribution de cette thèse a consisté en la conception (chapitre 3), la mise à l'épreuve (chapitre 4) et la préparation à l'extension aux ordres libres (chapitre 5) d'un formalisme de haut niveau, tourné vers l'ingénierie grammaticale, pour des tâches de description linguistique. Le formalisme et sa mise à l'épreuve ont fait l'objet d'une publication de journal [Clément *et al.*, 2015] ; et une version préliminaire des résultats du dernier chapitre a été présentée en conférence [Kirman et Salvati, 2013].

Résumé

Le formalisme en lui-même se compose de deux volets. D'une part, la description d'un langage de structures abstraites par l'intermédiaire d'une grammaire support (décrivant un langage d'arbres réguliers) agrémentée de contraintes de bonne formation (formulées dans un langage logique limité par la logique monadique du second ordre) permet de modéliser la structure syntaxique des énoncés, indépendamment de leur forme phonologique. D'autre part, un processus de linéarisation utilisant le lambda-calcul et guidé par la logique, utilisable en syntaxe comme en sémantique, permet de produire à partir de ces structures abstraites différents objets d'intérêt concret pour le TAL. Le lien entre ces objets et le langage abstrait est établi en s'appuyant sur des résultats existants dans la littérature des ACG et sur la connexion entre logique et automates, offrant une procédure de décision efficace à grammaire fixée – avec pour réserve que la taille des grammaires ainsi générées est en pratique excessive en l'absence de techniques d'optimisation spécifiques.

À l'usage, l'emploi de la logique pour définir des prédicats et relations additionnels permet de réifier des concepts linguistiques et de rendre explicites les hypothèses de modélisation de l'auteur d'une grammaire, facilitant sa maintenance ultérieure. Des réalisations phonologiques complexes, telles que l'ordre des mots en néerlandais, peuvent être décrites au moyen du lambda-calcul simplement typé, qui permet également des factorisations significatives en sémantique. Enfin, les phénomènes de mouvement et de dépendances à longue distance (illustrés par le mouvement *wh* et les contraintes d'îlot associées) peuvent être traités respectivement par l'emploi d'un mécanisme de requêtes

logiques et d'expression rationnelles intégrées au langage logique.

Enfin, l'outil algébrique de modélisation des ordres libres proposé dans le dernier chapitre induit une hiérarchie de grammaires commutatives, et nous a permis d'étudier leur complexité algorithmique en proposant des algorithmes d'analyse pour chacune des classes ainsi obtenues. Deux d'entre elles s'avèrent analysables efficacement à grammaire fixée, et permettent une modélisation simple des langues à ordre libre. Des phénomènes plus spécifiques, tels que le *scrambling*, peuvent également être modélisés de cette manière, bien que les langages résultants n'offrent pas de bonnes garanties de complexité algorithmique.

Questionnement

Au fil du document, nous avons soulevé plusieurs questions qui demeurent en suspens. Nous récapitulons ici quelques unes de ces interrogations, et tentons d'y apporter des éléments de réponse.

Compilation et optimisation La question la plus importante qui demeure non résolue à l'issue de ce travail est celle de la taille effective des grammaires générées par le formalisme. Nous nous sommes concentrés sur la complexité de l'analyse à grammaire fixée (qui est polynomiale) dans la majeure partie du document en supposant que, pour des applications de TAL, la grammaire est constante et peut donc être pré-calculée, optimisée et enregistrée à l'avance sur des supports de grande capacité. Cependant, il n'est pas exclu que la représentation compilée d'une grammaire crédible dépasse de loin les capacités de stockage des machines actuelles.

Le moyen le plus direct de répondre à cette question passe par l'implémentation du formalisme : celle-ci se heurtera tôt ou tard à ce problème, et requerra l'emploi de diverses techniques d'optimisation. Nous avons mentionné à l'issue du chapitre 3 l'existence d'outils logiciels dédiés à la vérification de modèles (*model-checking*), tels que MONA [Klarlund et Møller, 2001] ; ainsi que l'article de Morawietz et Cornell [1997], qui évoque plusieurs techniques d'optimisation possible pour la syntaxe en théorie des modèles. L'une d'elle est la construction d'automates partiels, indépendants, suivant une stratégie de type diviser-pour-régner ; une autre suppose la restriction de la grammaire a ses fragments applicables à l'énoncé à analyser – celle-ci entraîne cependant une compilation dynamique lors de l'analyse. Une solution modérée de ce type pourrait consister en la mise en cache de plusieurs grammaires « partielles », de taille raisonnable, et en le choix d'une ou plusieurs de ces grammaires précompilées en fonction du contenu de l'énoncé. Le cas échéant, il est aussi possible d'abandonner complètement l'étape de pré-compilation de la grammaire, en évaluant ses règles dynamiquement lors de l'analyse. À titre d'exemple, une

technique de ce type a été implémentée avec succès pour la vérification de certaines propriétés monadiques du second ordre sur les graphes par Courcelle et Durand [2011], sous la forme d'automates-à-la-volée (*fly-automata*).

Utilité des requêtes logiques L'un des composants les plus complexes du formalisme, compliquant considérablement le processus de compilation des linéarisations, est celui des requêtes logiques. Ces dernières ont été introduites pour faciliter le traitement du mouvement (notamment *wh*), et remplissent ce rôle efficacement ; toutefois, elles semblent constituer un ajout *ad-hoc* pour un phénomène linguistique spécifique. Le mouvement d'un fragment de réalisation peut également être traité par le lambda-calcul, en formant une paire de chaînes dont le premier élément est la chaîne localement réalisée, et le second est le composant déplacé (par exemple un pronom relatif). La réalisation du site d'insertion peut alors concaténer ces deux chaînes de la manière appropriée, comme le mentionne le paragraphe « Linéarisation alternative » page 92.

Un premier argument en leur faveur est cependant que, bien qu'elles compliquent l'implémentation du formalisme, l'emploi de ces requêtes facilite indiscutablement la modélisation du mouvement par rapport à l'emploi de paires de chaînes : or, la conception d'un outil de modélisation simple et de haut niveau reste notre objectif premier. Une autre raison de leur maintien dans le formalisme est l'existence d'autres emplois potentiels. D'une part, les phénomènes (particulièrement complexes) liés à l'ellipse pourraient bénéficier de la capacité de déplacer un composant commun dans la structure abstraite de la phrase, en fonction de certains choix de réalisation ; cette perspective semble toutefois assez incertaine. D'autre part, comme mentionné en conclusion du chapitre 4, les requêtes logiques pourraient être exploitées dans des linéarisations sémantiques pour assurer un traitement correct de la quantification sans requérir les constructions compositionnelles, à base de montée de type, actuellement dominantes dans la littérature.

Utilité de la grammaire support Une question plus subsidiaire dans la continuité de la précédente est celle de la nécessité de la grammaire support (ou grammaire d'approximation). Introduite pour faciliter la visualisation de la forme générale des structures abstraites et pour servir de point d'ancrage aux contraintes logiques et règles de linéarisation, elle n'est toutefois pas indispensable à la description des structures abstraites. Celles-ci pourraient en effet être entièrement spécifiées par des contraintes logiques. La perte en termes de simplicité visuelle semble trop importante pour justifier ce choix, mais il présente néanmoins un avantage : pour le rattachement de certaines contraintes à longue distance, telles que liens entre pronoms et antécédents ou entre site d'insertion et site d'extraction, le choix d'une production spécifique (ou plus) dans la grammaire support est quelque peu arbitraire. En un sens, ces contraintes

sont vraies à travers toute la structure abstraite, et non simplement là où une production les instancie.

Complexité du *scrambling* Une dernière interrogation porte sur la complexité algorithmique du *scrambling*. Comme souligné à l'issue du chapitre 5, celui-ci peut-être modélisé par des classes de grammaires commutatives dont la complexité est NP-difficile, et semble échapper à l'analyse par les classes polynomiales.

Il se pourrait cependant que cette difficulté soit intrinsèque au phénomène. L'exemple mentionné dans le chapitre 5 peut être assez rapidement résolu grâce au marquage des cas : les quatre arguments verbaux de la phrase ont autant de cas différents, le nominatif étant réservé à la première proposition dans une structure de contrôle, et chacun des trois compléments ne pouvant correspondre qu'à l'un des trois verbes présents. En l'absence de tels liens (ou de restrictions de sélection sémantique), la structure abstraite correspondant à un tel énoncé serait sous-spécifiée, permettant à une analyse quelconque d'être justifiable. Cependant, rien ne semble entraver en théorie la présence d'un grand nombre de restrictions grammaticales (ou sémantiques), limitant les possibilités d'appariement à certaines combinaisons de verbes et d'arguments ; dans un tel cas, la construction d'une structure abstraite satisfaisant toutes ces contraintes ressemble à la solution d'un problème de 3-couverture exacte, intrinsèquement NP-difficile.

Cet argument n'est toutefois pas entièrement satisfaisant, d'une part en raison de son côté très informel, et d'autre part parce que si le nombre de restrictions de sélection (ou de cas) est fini, ce qui est par hypothèse le cas dans une grammaire fixée, alors les regroupements effectués dans la structure abstraite (correspondant aux sous-ensembles de l'instance de X3C) possèdent une structure propre, susceptible d'altérer la complexité du problème.

Perspectives

Enfin, le formalisme tel qu'il est présenté dans le cadre de cette thèse ouvre de nombreuses perspectives de développement supplémentaires. La tâche la plus évidente est l'implémentation du formalisme, et son optimisation. Il s'agit cependant d'un travail considérable, impliquant la mise au point de nombreux composants logiciels inédits (par exemple pour l'analyse commutative) ou la réutilisation de bibliothèques (comme MONA) dont le développement est à l'arrêt. Le travail d'optimisation qui l'accompagne semble encore plus ambitieux, bien que sans doute plus prometteur sur le plan de l'intérêt des problèmes qu'il soulève.

Un autre travail, préalable, serait d'améliorer l'accessibilité du formalisme du point de vue d'un linguiste n'ayant pas d'expérience dans l'usage de la lo-

gique et du lambda-calcul. En effet, tout au long de ce document, nous n'avons pas particulièrement cherché à dissimuler le fonctionnement interne du formalisme (qui, au contraire, est au centre de la discussion) ; il paraît cependant peu concevable qu'un syntacticien souhaite s'embarrasser de lambda-abstractions, d'applications, et d'encodage des chaînes, termes, tuples et projections par le lambda-calcul simplement typé. Les primitives que représentent nos réalisations sont pourtant relativement simples : par exemple, les tuples de chaînes employés pour l'ordre des mots en néerlandais peuvent être plus aisément envisagés comme des « chaînes à trous » où les arguments et verbes s'insèrent de part et d'autre, et il en va de même des réalisations sémantiques, particulièrement complexes. Disposer d'une présentation plus accessible pour ces éléments (quitte à restreindre incidemment le pouvoir d'expression des linéarisations) pourrait faciliter l'abord de ces techniques de modélisation.

Un autre facteur de simplification des réalisations sémantiques serait, ainsi qu'évoqué précédemment, l'emploi de requêtes logiques pour simplifier le traitement de la sémantique. En effet, dans le paradigme actuel, la quantification porte sur une proposition complète, mais est décidée par les pronoms et déterminants de ses arguments. Pour remédier à ce problème, une construction à montée de type permet classiquement au déterminant de recevoir et de traiter, par le biais d'une continuation, le contenu de son groupe nominal, puis celui de l'ensemble de la phrase, ce qui alourdit inutilement le sens d'une proposition partielle. Une solution possible serait de requérir, à longue distance, une certaine propriété du déterminant (sélectionnant le type de quantification qu'il impose) et de construire la sémantique de toute la proposition d'une traite, simplifiant les types et formules intermédiaires. La clarté du résultat et la présence éventuelle d'obstacles à un tel traitement demeurent toutefois encore à déterminer.

Enfin, deux tâches restent à accomplir en rapport avec les grammaires commutatives du dernier chapitre. La première est leur intégration directe dans le formalisme dans le cadre d'une tâche de modélisation similaire à celle entreprise dans le chapitre 4. Cela permettrait de confronter à des phénomènes linguistiques réels notre représentation des ordres libres, et de garantir son bon fonctionnement. La seconde tâche consiste à explorer plus avant les propriétés de ces grammaires. Nous avons étudié en détail la complexité de l'analyse, mais certains détails restent en suspens : la complexité exacte de l'analyse universelle des CMCFG et CMG n'est pas fixée dans tous les cas, et les propriétés de clôture exactes des classes de langages commutatifs restent encore à établir.

Annexe A

Propriétés du système $\rightarrow_{\beta CE}$

A.1 Jonction des paires critiques (lemme 3.3)

Paires critiques de $\rightarrow_C$

1. $app, g/app, d$ – figure A.1 :
   — $app, g \rightarrow app, d \rightarrow cond$
   — $app, d \rightarrow app, g \rightarrow cond$ (modulo commutativité de $\wedge$)

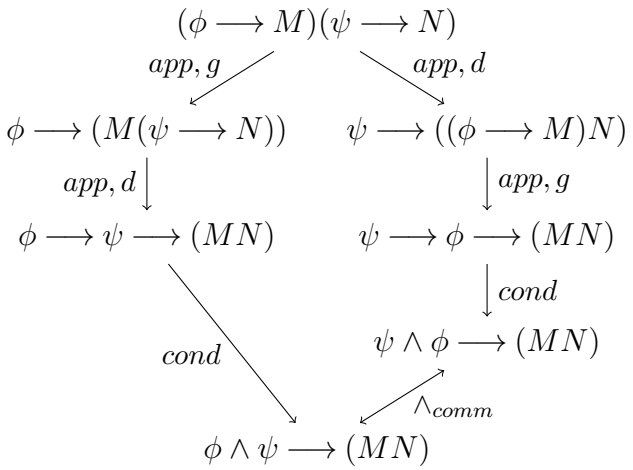


FIGURE A.1 – Résolution de $app, g/app, d$

2. $cond/r$ (avec $r \in \{abs; app, g; app, d; cond\}$) – figure A.2 :
   — $cond \to r$
   — $r \to r \to cond$ (si $r = cond$, on conclut par associativité de $\wedge$)

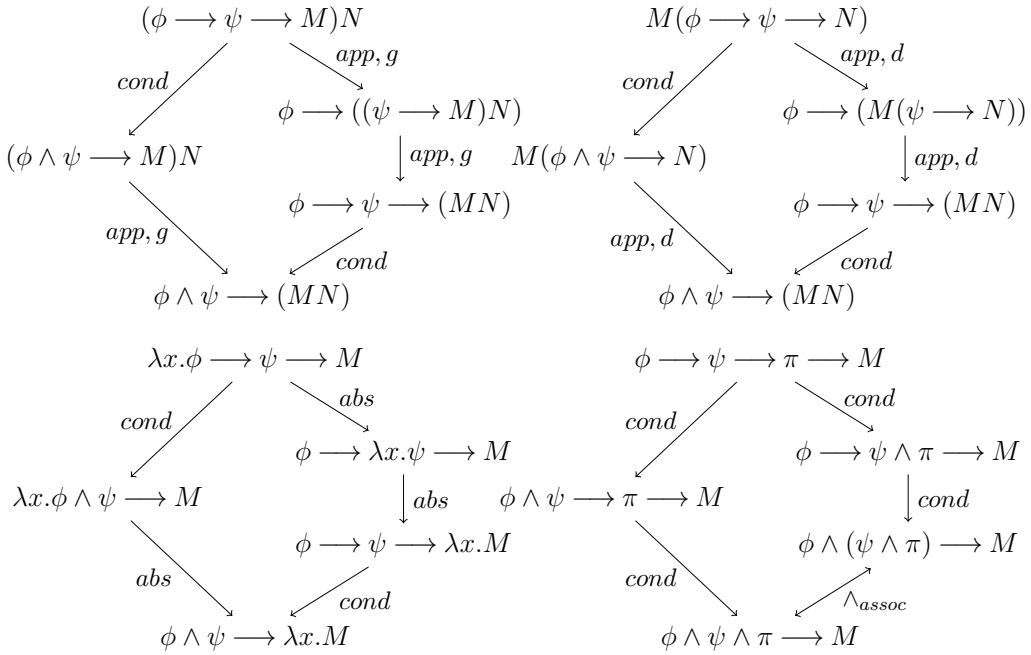


FIGURE A.2 – Résolution de $cond/r$

Paires critiques de $\to_E$

1. ens/r (avec $r \in \{app, g; app, d; abs; cond\}$ ou $r = ens$) – figure A.4 ou A.3 respectivement :
   — $ens \to r$
   — $r \to r \to ens$ (si $r = ens$, la dernière étape est exclue)

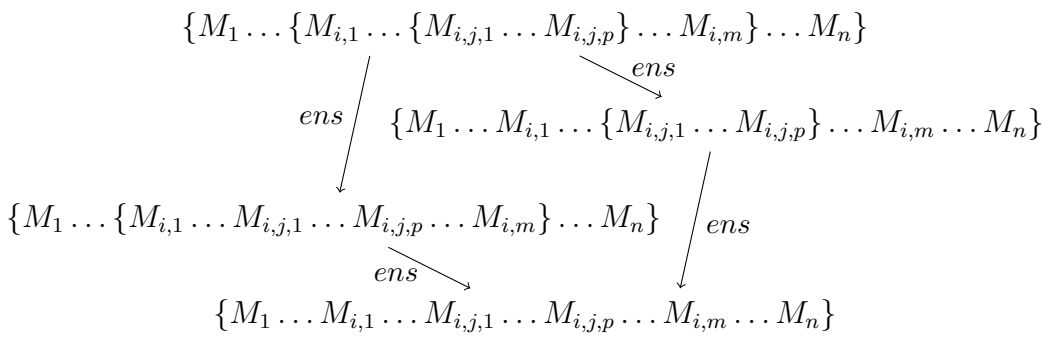


FIGURE A.3 – Résolution de ens/ens

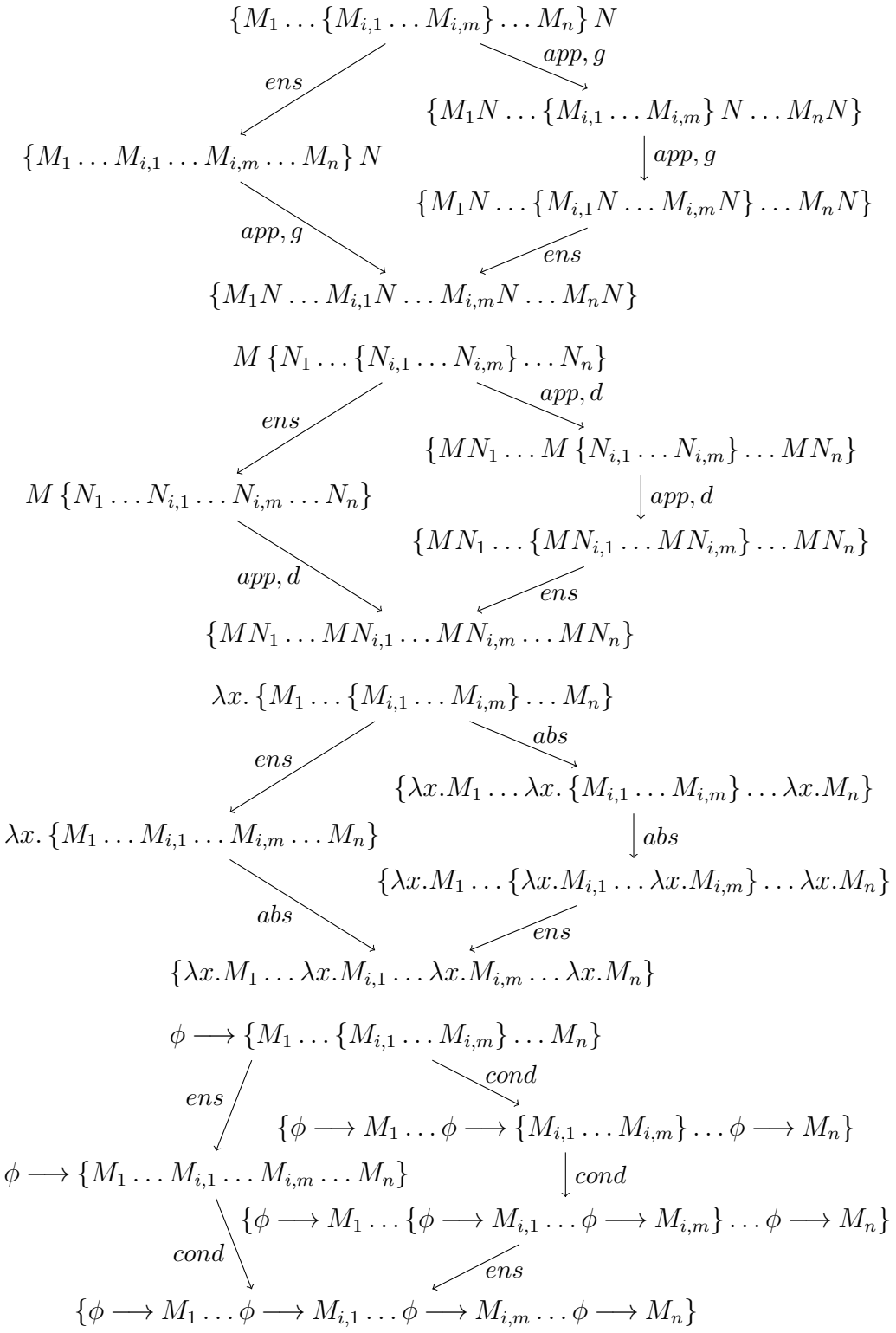


FIGURE A.4 – Résolution de ens/r ($r \neq ens$)

2. $app, g/app, d$ – figure A.5 :
   — $app, g(\to app, d)^n(\to ens)^n$
   — $app, d(\to app, g)^m(\to ens)^m$

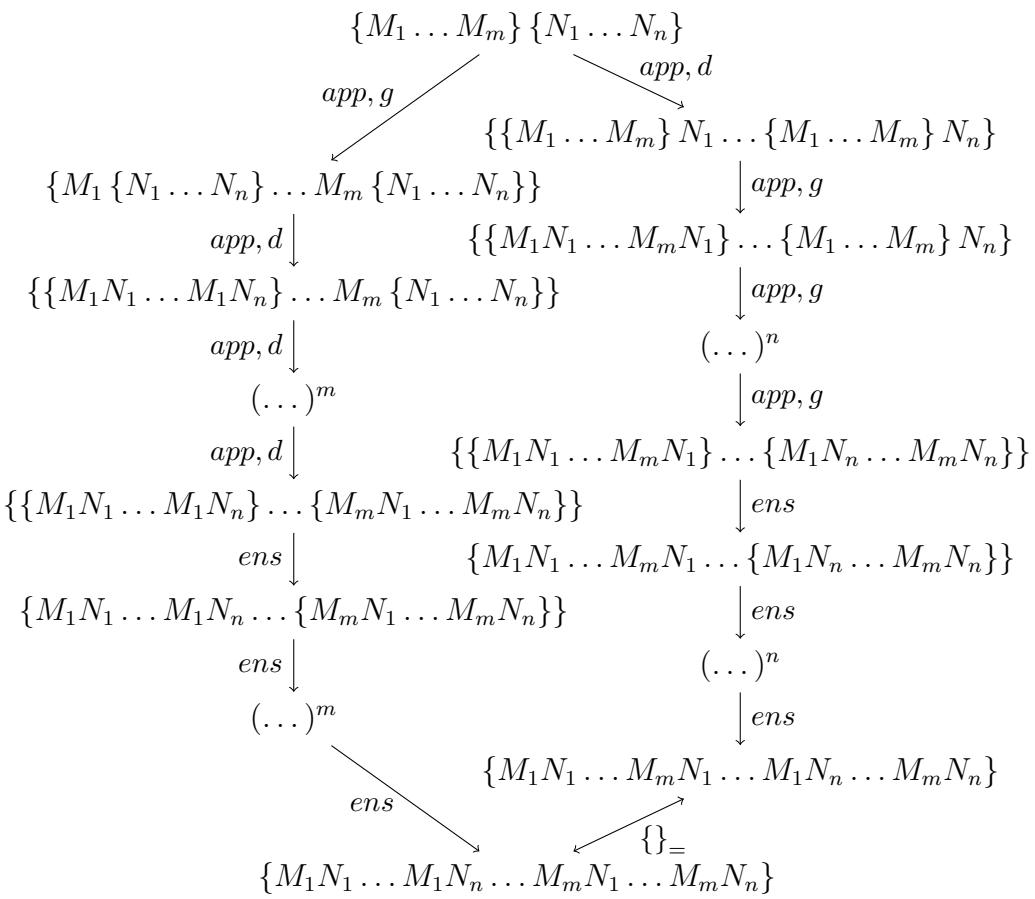


FIGURE A.5 – Résolution de $app, g/app, d$

3. sgl/r (avec $r \in \{app, g; app, d; abs; cond\}$ ou $r = ens$) – un exemple du premier cas ($cond$) est donné par la figure A.6, le second cas est trivial :
   — $r \rightarrow sgl$ (si $r = ens$, la paire est trivialement résolue sans sgl)
   — sgl

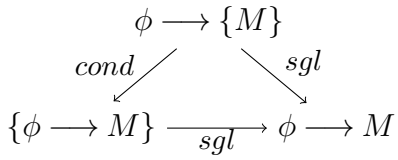


FIGURE A.6 – Résolution de sgl/r ($r = cond$)

Paires critiques de $\rightarrow_{CE}$

1. $cond_E/r_C$ (avec $r \in \{app, g; app, d; abs\}$ ou $r = cond$) – figure A.8 ou A.7 respectivement :
   — $cond_E \rightarrow r_E(\rightarrow r_C)^n$
   — $r_C \rightarrow r_E \rightarrow cond_E$ (la dernière étape est exclue si $r = cond$)

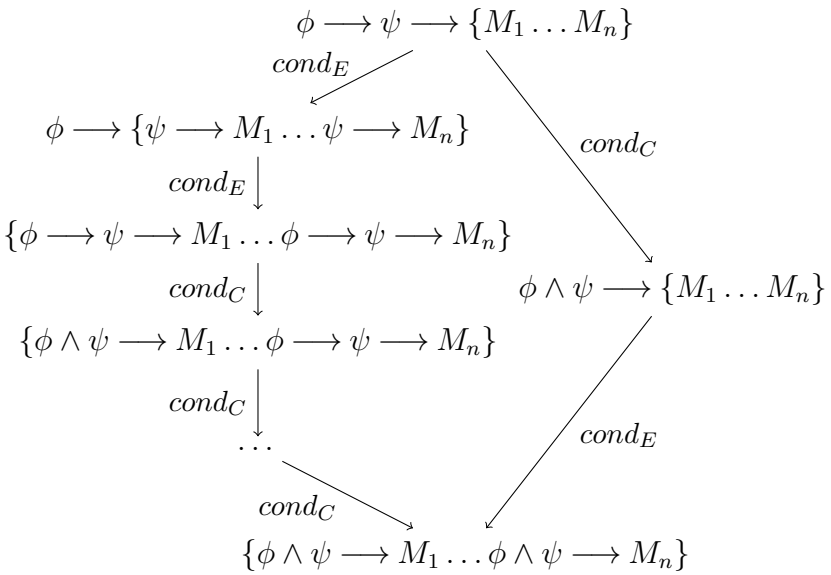


FIGURE A.7 – Résolution de $cond_E/cond_C$

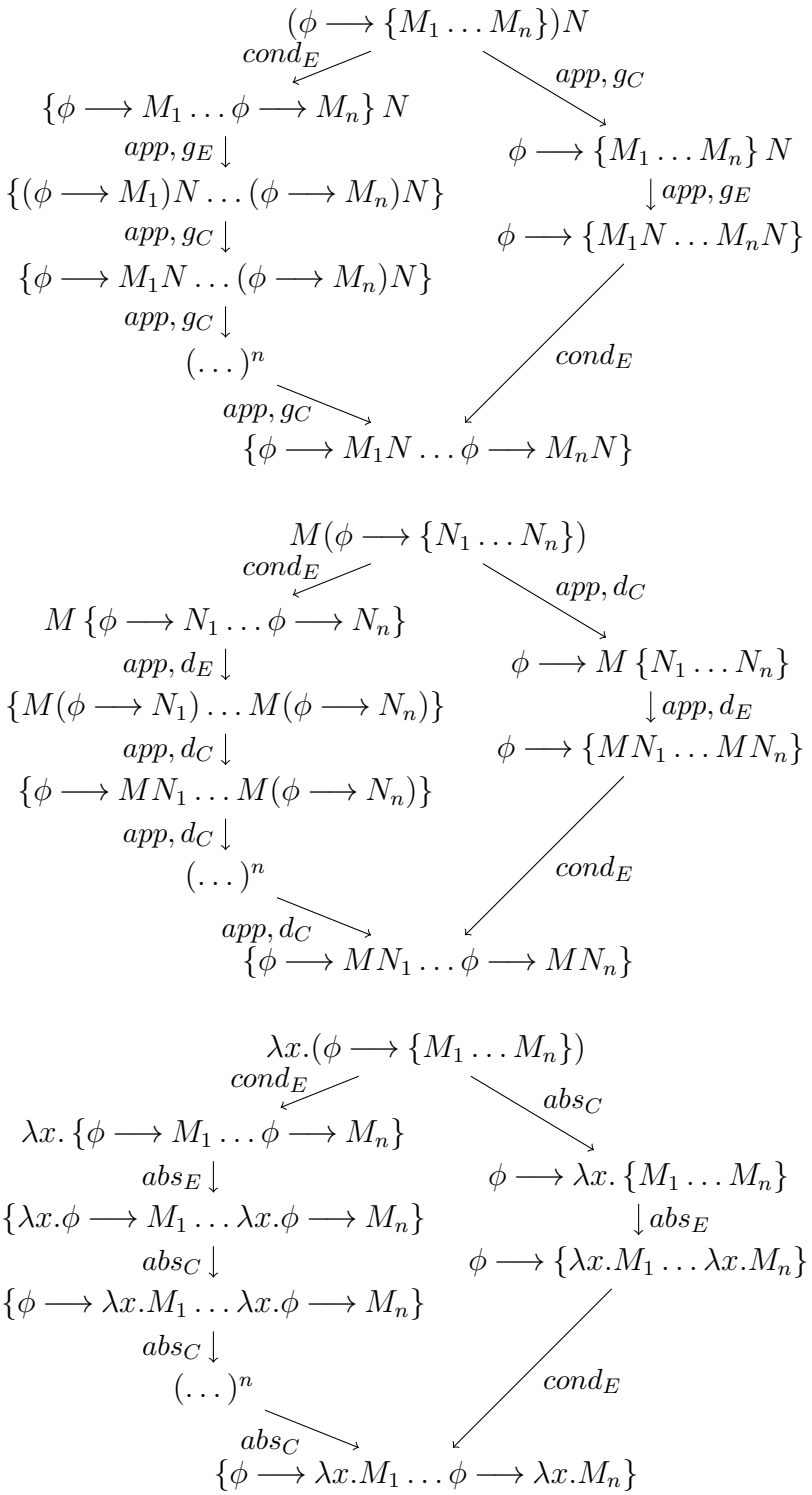


FIGURE A.8 – Résolution de $cond_E/r_C$ ($r \neq cond$)

2. $app, g_E/app, d_C$ – figure A.9 :
   — $app, g_E(\rightarrow app, d_C)^m$
   — $app, d_C \rightarrow app, g_E \rightarrow cond_E$

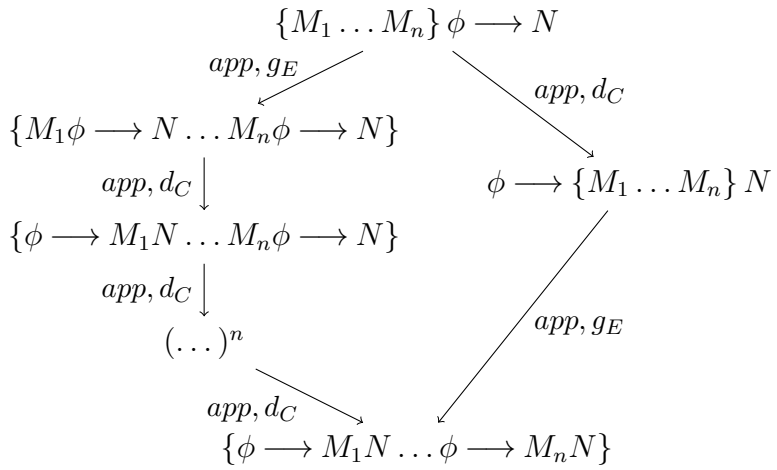


FIGURE A.9 – Résolution de $app, g_E/app, d_C$

3. $app, d_E/app, g_C$ – figure A.10 :
   — $app, d_E(\rightarrow app, g_C)^n$
   — $app, g_C \rightarrow app, d_E \rightarrow cond_E$

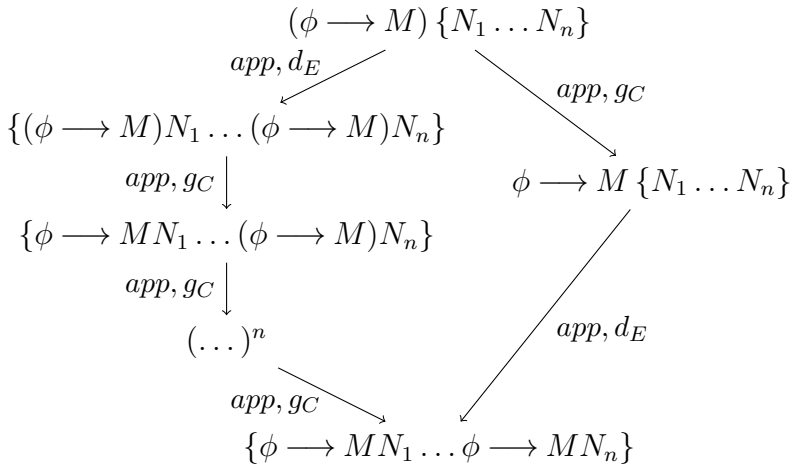


FIGURE A.10 – Résolution de $app, d_E/app, g_C$

Paires critiques de $\rightarrow_{\beta CE}$

1. β/abs_C :
   — β
   — $abs_C \rightarrow app, g \rightarrow \beta$

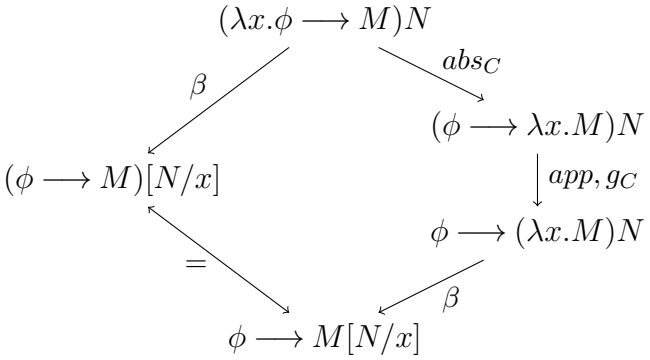


FIGURE A.11 – Résolution de β/abs_C

2. β/abs_E :
   — β
   — $abs_E \rightarrow app, g(\rightarrow \beta)^n$

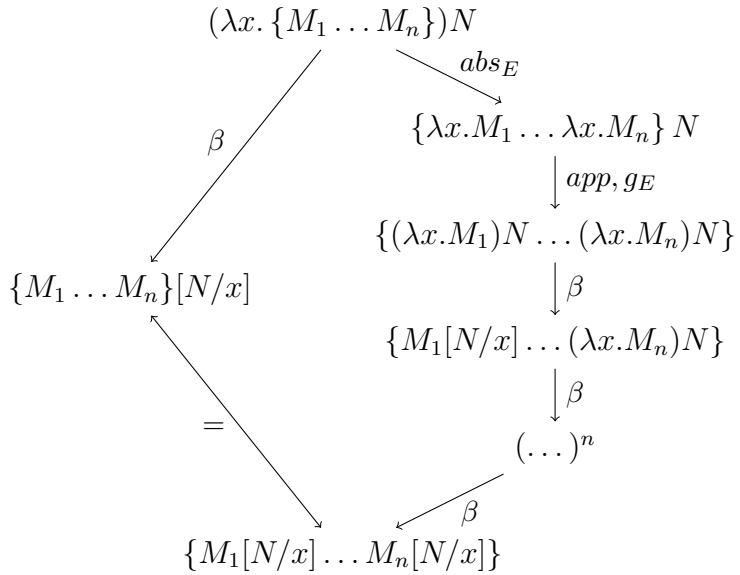


FIGURE A.12 – Résolution de β/abs_E

A.2 Normalisation forte (théorème 3.2)

Nous prouvons la normalisation forte du système $\rightarrow_{\beta CE}$ par la technique des candidats de réductibilité originellement due à Tait [1967]. La preuve suit le schéma donné dans Girard *et al.* [1989]ch. 6, en excluant le cas des paires, mais en incluant des hypothèses d'induction supplémentaires pour établir que la réductibilité d'un terme est, à l'image de son type, préservée par les constructions $\{\}$ et $\longrightarrow$. La troisième condition est également légèrement modifiée pour tenir compte de β-redex « latents » dans l'interprétation du terme (où l'abstraction est séparée de l'application par des ensembles ou des conditions).

A.2.1 Définitions

Nous introduisons tout d'abord les notions de termes réductibles, de termes neutres et de structure d'un terme. Dans le cadre de cette preuve, nous emploierons librement le mot *terme* pour désigner un lambda-terme étendu par des ensembles et des conditions, représentant une règle de linéarisation – un terme est essentiellement un élément de $\mathscr{R}$. En outre, nous utiliserons fréquemment dans nos notations les ensembles définis ici comme des prédicats : si E est un ensemble et M un terme, $E(M) \Leftrightarrow M \in E$. Sauf mention contraire, les notions de contraction/expansion/réduction recouvriront implicitement celles issues de la relation $\rightarrow_{\beta CE}$. Enfin, FN est l'ensemble des termes fortement normalisants (pour lesquels il n'existe aucune séquence infinie de réductions) ; et $\nu(M)$ est la longueur maximale d'une réduction dans $M \in \mathsf{FN}$.

Définition A.1. Pour tout type simple τ, l'ensemble R_τ est l'ensemble des termes *réductibles*, défini comme suit :

- Si $\tau = \alpha$ est un type atomique, R_α est l'ensemble des termes de type α qui sont fortement normalisants.
- Si $\tau = \alpha \rightarrow \beta$ est un type simple, $\mathsf{R}_{\alpha\rightarrow\beta}$ est l'ensemble des termes M de type $\alpha \rightarrow \beta$ tels que pour tout terme N appartenant à R_α, le terme MN appartient à R_β.

Définition A.2. Un terme est dit *neutre* si et seulement si son application à un autre terme ne permet pas de créer de β-redex supplémentaire.

En d'autres termes, un terme est neutre si et seulement si il est une constante, une variable, une application, une condition portant sur un terme neutre ou un ensemble de termes neutres. L'ensemble des termes neutres est noté Nt, et défini comme suit :

- Si x est une variable, alors $\mathsf{Nt}(x)$.
- Si c est une constante, alors $\mathsf{Nt}(c)$.
- Si M et N sont des termes (quelconques), alors $\mathsf{Nt}(MN)$.
- Si M est un terme neutre et ϕ une formule logique, alors $\mathsf{Nt}(\phi \longrightarrow M)$.
- Si $M_1 \dots M_n$ sont des termes neutres, alors $\mathsf{Nt}(\{M_1 \dots M_n\})$.

Définition A.3. La *structure* $\mathsf{S}(M)$ d'un terme M désigne l'ensemble des sous-termes de M dominés uniquement par des conditions ou des ensembles :

— $\mathsf{S}(M) = \{M\} \cup \bigcup_{i\in[n]} \mathsf{S}(M_i)$ si $M = \{M_1 \dots M_n\}$.
— $\mathsf{S}(M) = \{M\} \cup \mathsf{S}(M')$ si $M = \{\phi \longrightarrow M'\}$.
— $\mathsf{S}(M) = \{M\}$ dans tous les autres cas.

Nous introduisons maintenant une notion de vérification *locale* d'une propriété sur nos termes. Pour toute propriété des termes, cette propriété est dite localement vérifiée dans un terme M si et seulement si elle est vérifiée par tous les éléments minimaux de sa structure $\mathsf{S}(M)$ – c'est-à-dire les éléments de $\mathsf{S}(M)$ qui ne sont ni des ensembles ni des conditions.

Définition A.4. Soit $P(M)$ un prédicat sur les termes. Le prédicat $P_\downarrow(M)$, signifiant que M *vérifie localement* P, est défini comme suit :

$$P_\downarrow(M) \overset{def}{\Leftrightarrow} \begin{cases} \forall i \in [n].P_\downarrow(M_i) & \text{si } M = \{M_1 \dots M_n\} \\ P_\downarrow(M_0) & \text{si } M = \phi \longrightarrow M_0 \\ P(M) & \text{sinon} \end{cases}$$

Nous montrons maintenant que, selon cette notion, la normalisation forte d'un terme est une conséquence directe de la normalisation forte locale des termes de sa structure. Ce lemme nous permettra d'établir par la suite que les ensembles et conditions que nous avons introduits dans nos règles de linéarisation préservent la normalisation forte des termes et, par suite, leur réductibilité.

Lemme A.1. *Pour tout terme M, si M est localement fortement normalisant, alors M est fortement normalisant* ($\mathsf{FN}_\downarrow(M) \Rightarrow \mathsf{FN}(M)$).

Démonstration. Ce lemme se démontre par induction sur la structure de M.

Si M n'est ni un ensemble ni une condition, alors $\mathsf{FN}_\downarrow(M) \overset{def}{\Leftrightarrow} \mathsf{FN}(M)$.

Nous considérons maintenant le cas où M est un ensemble. Dans ce cas, $M = \{M_1 \dots M_n\}$ et, par hypothèse d'induction, $\forall i \in [n].\, \mathsf{FN}(M_i)$. Nous montrons que $M \in \mathsf{FN}$ par induction sur $\sum \nu(M_i)$ et $|M|$. Si M est en forme normale, $\mathsf{FN}(M)$ est immédiatement vérifié. Dans le cas contraire, nous avons $M \to M'$ avec l'un des cas suivants :

— Soit $M' = \{M_1 \dots M'_i \dots M_n\}$ et $M_i \to M'_i$. Dans ce cas, $\nu(M'_i) < \nu(M_i)$ et M' est fortement normalisant par hypothèse d'induction, nous permettant de conclure.
— Soit $M' = M_1$ par application de *sgl*, avec $n = 1$. Dans ce cas, M_1 est fortement normalisant et nous pouvons conclure.
— Soit $M' = \{M_1 \dots M_{k,1} \dots M_{k,m} \dots M_n\}$ par application de *ens*, avec $M_k = \{M_{k,1} \dots M_{k,m}\}$. Dans ce cas, la somme des $\nu(M_j)$ pour $M_j \in M'$ est inférieure ou égale à la somme des $\nu(M_i)$ pour $M_i \in M$, puisque les termes contenus dans M_k sont normalisables en moins de $\nu(M_k)$ étapes. En outre, $|M'| < |M|$; donc, par hypothèse d'induction, $\mathsf{FN}(M')$, entraînant le résultat.

Par conséquent, nous établissons que tout ensemble de termes fortement normalisants est fortement normalisant.

Enfin, considérons le cas où M est une condition, avec $M = \phi \longrightarrow M_0$. Comme dans le cas précédent, nous montrons alors que $M \in \mathsf{FN}$ par induction sur $\nu(M_0)$ et $|M|$, le cas de base où M est en forme normale étant trivialement vérifié. Supposons donc que $M \rightarrow M'$, trois types de contractions sont possibles :

- Soit le redex contracté dans M' appartient à M_0 et $M' = \phi \longrightarrow M'_0$. Alors $\nu(M'_0) < \nu(M_0)$, et M' est fortement normalisant par hypothèse d'induction, validant ce cas.
- Soit $M' = \phi \wedge \psi \longrightarrow N$, avec $M_0 = \psi \longrightarrow N$. Dans ce cas, $|M'| < |M|$ avec $\nu(N) \leq \nu(M_0)$; M' est alors fortement normalisant par hypothèse d'induction (le fait de substituer $\phi \wedge \psi$ à ϕ n'affectant aucune des règles de réduction existantes).
- Soit enfin $M' = \{\phi \longrightarrow M_1 \ldots \phi \longrightarrow M_n\}$ car $M_0 = \{M_1 \ldots M_n\}$. Dans ce dernier cas, pour tout $i \in [n]$, $|M_i| < |\{M_1 \ldots M_n\}|$ et $\nu(M_i) \leq \nu(\{M_1 \ldots M_n\})$, donc, par hypothèse d'induction, $\phi \longrightarrow M_i$ est fortement normalisant pour tout $i \in [n]$. Or, nous avons montré précédemment que tout ensemble de termes fortement normalisants est fortement normalisant, permettant de déduire là encore que $M' \in \mathsf{FN}$, concluant le lemme. □

A.2.2 Propriétés des ensembles réductibles

Nous établissons maintenant un ensemble de propriétés des ensembles réductibles R_τ, que nous démontrons inductivement pour tout type τ. Outre les propriétés **(CR 1)** à **(CR 4)** issues de la preuve de normalisation citée plus haut, nous montrons par **(CR 5)** et **(CR 6)** qu'un terme est réductible si et seulement si il est localement réductible ($\mathsf{R}_\tau \Leftrightarrow \mathsf{R}_{\tau\downarrow}$).

(CR 1) $\mathsf{R}_\tau(M) \Rightarrow \mathsf{FN}(M)$: tout terme qui est réductible est également fortement normalisant.

(CR 2) $\mathsf{R}_\tau(M) \wedge M \stackrel{*}{\rightarrow} M' \Rightarrow \mathsf{R}_\tau(M')$: la réductibilité d'un terme M est préservée par réduction.

(CR 3) $\mathsf{Nt}(M) \wedge (\forall M_i \in \mathsf{S}(M).M_i \rightarrow M'_i \Rightarrow \mathsf{R}_\tau(M'_i)) \Rightarrow \mathsf{R}_\tau(M)$: si un terme M est neutre, et que tout élément M_i de sa structure se contracte en un terme M'_i réductible de type τ, alors le terme d'origine (M) est également réductible.

(CR 4) $\mathsf{Nt}(M) \wedge (\nexists M'.M \rightarrow M') \Rightarrow \mathsf{R}_\tau$: tout terme M de type τ qui est neutre et en forme normale est réductible.

(CR 5) $\mathsf{R}_\tau(M) \Rightarrow \mathsf{R}_{\tau\downarrow}(M)$: un terme réductible est localement réductible.

(CR 6) $\mathsf{R}_{\tau\downarrow}(M) \Rightarrow \mathsf{R}_\tau(M)$: un terme localement réductible est réductible.

La propriété **(CR 4)** est une conséquence immédiate de **(CR 3)** : le fait que M soit en forme normale ($\nexists M'.M \rightarrow M'$) satisfait trivialement la seconde

précondition de **(CR 3)**. Nous démontrons pour tout terme M la validité des autres propriétés par induction sur le type τ de M.

Dans le cas où τ est un type atomique :

(CR 1) est vraie par définition : $\mathsf{R}_\tau(M) \overset{def}{\Leftrightarrow} \mathsf{FN}(M)$.

(CR 2) est immédiatement vérifiée par définition, toute réduction d'un terme de FN produisant un terme de FN.

(CR 3) se vérifie en prenant $M_i = M$: la deuxième précondition entraîne alors que tout contractum de M est fortement normalisant, donc $M \in \mathsf{FN}$ également.

(CR 5) est vérifiée en constatant que toute séquence de réductions infinie dans un terme local M_i appartenant à $\mathsf{S}(M)$ serait également valide dans M, contredisant l'hypothèse $\mathsf{R}_\tau(M)$.

(CR 6) est par définition, l'énoncé du lemme A.1.

Nous étudions maintenant le cas inductif où $\tau = \alpha \to \beta$: nous disposons des six hypothèses d'induction précédentes, appliquées aux types α et β. Rappelons que par définition, un terme M appartient à $\mathsf{R}_{\alpha\to\beta}$ si et seulement si pour tout terme N réductible de type α, MN est un terme réductible (de type β) :

(CR 1) Par définition, $\mathsf{R}_{\alpha\to\beta}(M)$ entraîne que pour tout N réductible de type α, $MN \in \mathsf{R}_\beta$. En particulier, $Mx^\alpha \in \mathsf{R}_\beta$, puisque $x^\alpha \in \mathsf{R}_\alpha$ en vertu de la propriété **(CR 4)** pour α (x est neutre et en forme normale). Par conséquent, $Mx \in \mathsf{FN}$, en raison de l'hypothèse d'induction **(CR 1)** pour β ; ce qui entraîne en particulier que $M \in \mathsf{FN}$.

(CR 2) Là encore, tout $N \in \mathsf{R}_\alpha$ vérifie $MN \in \mathsf{R}_\beta$; de plus, $MN \overset{*}{\to} M'N$. L'hypothèse d'induction **(CR 2)** sur β permet donc d'établir que $M'N \in \mathsf{R}_\beta$ pour tout $N \in \mathsf{R}_\alpha$ et donc, par définition, que $M' \in \mathsf{R}_\tau$.

(CR 3) Posons $\mathsf{Nt}(M)$ et $\mathbf{H} \overset{def}{\Leftrightarrow} \forall M_i \in \mathsf{S}(M). M_i \to M_i' \Rightarrow \mathsf{R}_\tau(M_i')$.
Nous souhaitons montrer que $M \in \mathsf{R}_\tau$, c'est-à-dire, par définition, que pour tout $N \in \mathsf{R}_\alpha$, $MN \in \mathsf{R}_\beta$. Par induction, la propriété **(CR 3)** pour le type β permet d'obtenir ce résultat en montrant que MN est neutre (ce qui est toujours vrai par définition de Nt), et que pour tout N, si un élément O de la structure de MN se contracte en P, alors P appartient à R_β. Par définition, la structure de MN est simplement $\{MN\}$; la condition précédente se réduit ainsi à $MN \to P \Rightarrow \mathsf{R}_\beta(P)$. Nous démontrons donc dans la suite, pour chaque cas, que $\forall N \in \mathsf{R}_\alpha . MN \to P \Rightarrow \mathsf{R}_\beta(P)$, et concluons que $M \in \mathsf{R}_\tau$ de la manière décrite ci-dessus.
Tout terme réductible N^α est fortement normalisant, par **(CR 1)** sur α. Supposons que MN se contracte en P (dans le cas contraire, MN est neutre et normal, et nous pouvons conclure immédiatement par **(CR 4)**) ; six cas sont alors possibles. La contraction peut avoir lieu dans M ou N, auquel cas $P = M'N$ avec $M \to M'$ (cas 1), ou $P = MN'$ avec $N \to N'$ (cas 2). Alternativement, lorsque M ou N est un ensemble ou une condition, soit $P = \phi \longrightarrow M_0 \wedge M = \phi \longrightarrow M_0$ (cas 3), soit $P = \{M_1 N \dots M_n N\} \wedge M = \{M_1 \dots M_n\}$ (cas 4), soit $P = MN_0 \wedge N =$

$\phi \longrightarrow N_0$ (cas 5), soit enfin $P = \{MN_1 \dots MN_n\} \wedge N = \{N_1 \dots N_n\}$ (cas 6). Observons que M étant neutre, MN ne peut pas constituer un β-redex.

Nous montrons maintenant que $P \in \mathsf{R}_\beta$ par une induction complexe sur MN : nous montrons que la propriété est vérifiée pour tout terme $M'N'$, dans lequel : soit $|M'| < |M|$, soit $|N'| < |N|$ ou $\nu(N') < \nu(N)$ (il est possible dans ce dernier cas que $|N'| > |N|$, mais l'inverse n'est pas vrai). La première condition de décroissance correspond aux cas 3 et 4 (induction sur la structure de M), la seconde aux cas 5 et 6 (induction sur la structure de N), et la dernière au cas 2 (en s'appuyant sur l'argument montrant que $N \in \mathsf{FN}$) ; le cas 1 est démontré sans faire usage de l'hypothèse d'induction. Nous montrons dans chaque cas que P (le contractum de MN) est réductible ; la réductibilité de MN (qui permet le passage à l'induction) est toujours une conséquence de celle de P, suivant le raisonnement décrit dans le premier paragraphe.

1. $P = M'N$ $\qquad (M \rightarrow M')$

   Considérons l'hypothèse **H** posée initialement, fournie par **(CR 3)**, en prenant $M_i \in \mathsf{S}(M) = M$: elle établit que $M' \in \mathsf{R}_\tau$. Donc, par définition, $M'N = P \in \mathsf{R}_\beta$ pour tout $N \in \mathsf{R}_\alpha$.

2. $P = MN'$ $\qquad (N \rightarrow N')$

   Puisque $N \rightarrow N'$, $\nu(N') < \nu(N)$, et la propriété **(CR 2)** sur α permet d'établir que $\mathsf{R}_\alpha(N')$. Nous pouvons donc appliquer notre hypothèse d'induction pour établir immédiatement que $P = MN'$ appartient à R_β.

3. $P = \phi \longrightarrow M_0N$ $\qquad (M = \phi \longrightarrow M_0)$

   Dans ce cas, $|M_0| < |M|$, donc par hypothèse d'induction, M_0N est réductible de type β. La propriété **(CR 6)** pour β permet donc immédiatement d'établir $\mathsf{R}_\beta(P)$.

4. $P = \{M_1N \dots M_nN\}$ $\qquad (M = \{M_1 \dots M_n\})$

   De manière similaire au cas précédent, pour tout $i \in [n]$, $|M_i| < |M|$, donc par hypothèse d'induction, M_iN est réductible de type β, et la propriété **(CR 6)** pour β permet alors d'établir $\mathsf{R}_\beta(P)$.

5. $P = \phi \longrightarrow MN_0$ $\qquad (N = \phi \longrightarrow N_0)$

   Ici, $|N_0| < |N|$; en outre, la propriété **(CR 5)** montre $\mathsf{R}_\alpha(N_0)$. Nous pouvons donc appliquer notre hypothèse d'induction pour établir $\mathsf{R}_\beta(MN_0)$, ce qui entraîne $\mathsf{R}_\beta(P)$ par la propriété **(CR 6)** de β.

6. $P = \{MN_1 \dots MN_n\}$ $\qquad (N = \{N_1 \dots N_n\})$

   Similairement au cas précédent, nous montrons que pour tout $i \in [n]$, $|N_i| < |N|$ et $\mathsf{R}_\alpha(N_i)$ (par **(CR 5)**). Par conséquent, chaque MN_i est réductible de type β, montrant par **(CR 6)** sur β que $\mathsf{R}_\beta(P)$.

Nous avons donc établi que pour tout $N \in \mathsf{R}_\alpha$, dans tous les cas, si $MN \to P$ alors $P \in \mathsf{R}_\beta$. Nous pouvons donc conclure systématiquement en suivant le raisonnement du premier paragraphe pour obtenir que $MN \in \mathsf{R}_\beta$ par **(CR 3)** sur le type β (validant pour chaque cas l'induction sur M, N et $\nu(N)$), et par extension que $M \in \mathsf{R}_\tau$, vérifiant ainsi la propriété **(CR 3)** pour τ.

(CR 5) Nous montrons la réductibilité locale par induction sur M. Si M n'est ni un ensemble ni une condition, la propriété est vraie par définition.

Si M est une condition $\phi \longrightarrow M_0$ réductible de type τ, alors (par définition) pour tout $N \in \mathsf{R}_\alpha$, $\mathsf{R}_\beta((\phi \longrightarrow M_0)N)$. Donc, par **(CR 2)** sur β, $\mathsf{R}_\beta(\phi \longrightarrow M_0N)$ (en appliquant la règle app, g de $\to_C$), ce qui implique à son tour par **(CR 5)** sur le type β que $\mathsf{R}_\beta(M_0N)$ pour tout $N \in \mathsf{R}_\alpha$, et donc, par définition, $\mathsf{R}_\tau(M_0)$.

Si M est un ensemble, la preuve suit le même schéma que dans le cas précédent, avec $M = \{M_1 \dots M_n\}$ et en appliquant la règle app, g de $\to_E$. Il en résulte que pour tout $i \in [n]$, $M_iN \in \mathsf{R}_\beta$ pour tout N réductible de type α, montrant que $M_i \in \mathsf{R}_\tau$.

(CR 6) Considérons le terme M : si M n'est pas un ensemble ou une condition, la propriété est une tautologie. Nous supposons donc le contraire dans la suite, et montrons que M est réductible.

Par définition, M est réductible si, pour tout $N \in \mathsf{R}_\alpha$, $\mathsf{R}_\beta(MN)$. Nous montrons ce dernier résultat en utilisant **(CR 3)** sur le type β, et en montrant que tout contractum du terme neutre MN est réductible, par une induction complexe sur MN.

Nous observons d'abord que la réductibilité locale de M entraîne par **(CR 1)** appliquée au type τ que M est localement fortement normalisant : en effet, tous les termes de $\mathsf{S}(M)$ ont le type τ, et la propriété **(CR 1)** est établie pour τ. Par conséquent, d'après le lemme A.1, M est fortement normalisant, nous permettant de procéder par induction sur $\nu(M)$.

Par hypothèse d'induction, nous supposons maintenant que $\mathsf{R}_\beta(M'N')$ soit lorsque $\nu(M') < \nu(M)$ ou, à défaut, $|M'| < |M|$ (sans accroître $\nu(M')$), soit lorsque $\nu(N') < \nu(N)$ ou $|N'| < |N|$. Nous considérons tous les termes P tels que $MN \to P$ et montrons que P appartient systématiquement à R_β, établissant par **(CR 3)** que $MN \in \mathsf{R}_\beta$ et propageant l'induction. (Le terme MN est forcément réductible, de par notre supposition initiale que M est soit un ensemble soit une condition.)

Nous considérons d'abord les réductions qui sont dues au terme N, indépendamment de M. Remarquons que la réductibilité de N entraîne par **(CR 2)** sur le type α que tout N' tel que $N \xrightarrow{*} N'$ est réductible, et par **(CR 5)** (sur α) que tout $N_i \in \mathsf{S}(N)$ est également réductible.

Trois cas sont à considérer :
— Soit $P = MN'$ avec $N \rightarrow N'$. Dans ce cas, P est réductible par hypothèse d'induction, avec $\nu(N') < \nu(N)$.
— Soit $P = \psi \rightarrow MN_0$ avec $N = \psi \longrightarrow N_0$. Dans ce cas, MN_0 est réductible par hypothèse d'induction, avec $|N_0| < |N|$; ce qui entraîne que P est réductible par application de **(CR 6)** sur β.
— Soit $P = \{MN_1 \dots MN_n\}$ avec $N = \{N_1 \dots N_n\}$. Dans ce cas, chaque MN_i est réductible par hypothèse d'induction (avec $|N_i| < |N|$) ; donc P est à nouveau réductible par **(CR 6)** sur β.

Nous considérons ensuite les réductions dues à M seulement. Ceci recouvre tous les cas restants : MN ne peut être un β-redex, puisque nous avons écarté initialement les cas où M n'est ni une condition, ni un ensemble.
— Soit $P = M'N$ avec $M \rightarrow M'$. Dans ce cas, P est réductible par hypothèse d'induction, avec $\nu(M') < \nu(M)$.
— Soit $P = \phi \longrightarrow M_0N$, avec $M = \phi \longrightarrow M_0$. Dans ce cas, M_0N est réductible par hypothèse d'induction, avec $|M_0| < |M|$; donc P est réductible en appliquant la propriété **(CR 6)** sur le type β.
— Soit $P = \{M_1N \dots M_nN\}$, avec $M = \{M_1 \dots M_n\}$. Dans ce cas, chaque M_iN est réductible par hypothèse d'induction, avec $|M_i| < |M|$; donc, P est encore une fois réductible par **(CR 6)** sur β, concluant la démonstration. □

A.2.3 Réductibilité des termes de $\mathscr{R}$

Nous souhaitons maintenant montrer que tous les termes de $\mathcal{R}$ sont réductibles et, par extension, fortement normalisants. Dans ce but, nous montrons au préalable un lemme supplémentaire qui nous permettra de traiter le cas des abstractions.

Lemme A.2. *Si $M[P/y]$ est réductible pour tout terme P réductible, alors $\lambda y.M$ est réductible.*

Démonstration. Soit α le type de y et β celui de M. Nous montrons que que $\mathsf{R}_{\alpha\rightarrow\beta}(\lambda y.M)$ par induction sur la structure de M – nous notons **H1** l'hypothèse d'induction correspondante, à savoir que si $M = \phi \longrightarrow M_0$ ou $M = \{M_1 \dots M_n\}$ alors $\lambda y.M_i \in \mathsf{R}_{\alpha\rightarrow\beta}$ pour tout $i \in [n]$.

Observons que l'hypothèse du lemme est préservée par cette induction : si $M[P/y]$ est réductible et que M est un ensemble ou une condition, alors pour tout sous-terme M_i de M, $M_i[P/y]$ est réductible par **(CR 5)**.

Par définition, $\lambda y.M$ est réductible si pour tout N réductible, $(\lambda y.M)N$ est réductible. Posons $R = (\lambda y.M)N$, et montrons que ce terme est réductible : R est neutre et $\mathsf{S}(R) = \{R\}$, donc, par **(CR 3)**, si $R \rightarrow R'$ et $\mathsf{R}_{\alpha\rightarrow\beta}(R')$, alors R est réductible, et le lemme est vérifié.

Le terme N est réductible, et donc, par **(CR 4)**, fortement normalisant. Par l'hypothèse du lemme, M est également réductible (en prenant $P = y$) et donc fortement normalisant. Nous montrons maintenant que tout R' est réductible par induction : l'hypothèse correspondante **H2** est que $(\lambda y.M')N'$ est réductible soit lorsque $\nu(M') < \nu(M)$, soit lorsque $(\nu(N'), |N'|) < (\nu(N), |N|)$ (dans l'ordre lexicographique).

Nous considérons maintenant les valeurs possibles de R' :

— $R' = M[N/y]$. Ce terme est réductible par l'hypothèse du lemme.
— $R' = (\lambda y.M')N$ avec $\nu(M') < \nu(M)$. Alors, R' est réductible par **H2**.
— $R' = (\lambda y.M)N'$ avec $\nu(N') < \nu(N)$. Là aussi, R' est réductible par **H2**.
— $R' = \{(\lambda x.M)N_1 \dots (\lambda x.M)N_n\}$ avec $|N_i| < |N|$ et $\nu(N_i) \leq \nu(N)$ pour tout $i \in [n]$. Dans ce cas, chaque $(\lambda x.M)N_i$ est réductible par **H2** ; donc, R' est réductible par **(CR 6)**.
— $R' = \phi \longrightarrow (\lambda x.M)N_0$ avec $|N_0| < |N|$ et $\nu(N_0) \leq \nu(N)$. Là encore, la réductibilité de R' se montre par application de **H2** et **(CR 6)**.
— $R' = \{\lambda x.M_1 \dots \lambda x.M_n\}\, N$, avec $|M_i| < |M|$ pour tout $i \in [n]$. Donc, par **H1**, $\lambda x.M_i$ est réductible pour tout i, entraînant par **(CR 6)** que l'ensemble $\{\lambda x.M_1 \dots \lambda x.M_n\}$ est réductible. Par définition de la réductibilité, l'application R' de cet ensemble réductible à un terme N réductible est elle-même réductible.
— $R' = (\phi \longrightarrow M_0)N$ avec $|M_0| < |M|$. Suivant le même raisonnement, R' est réductible par application de **H1**, **(CR 6)** et par définition de la réductibilité. □

Nous montrons maintenant que tous les termes sont réductibles ; par suite, la propriété **(CR 1)** entraîne immédiatement la normalisation forte du système $(\mathscr{R}, \to_{\beta CE})$.

Théorème A.3. *Tout terme $M \in \mathscr{R}$ est réductible.*

Corollaire A.4. *Tout terme $M \in \mathscr{R}$ est fortement normalisant.*

Démonstration. Comme pour le cas du lambda-calcul simplement typé, la preuve procède en montrant par induction structurelle que pour tout terme M dont les variables libres sont $\mathrm{FV}(M) = \{x_1^{\alpha_1} \dots x_m^{\alpha_m}\}$, si $P_1 \dots P_m$ sont des termes appartenant respectivement à $\mathsf{R}_{\alpha_1} \dots \mathsf{R}_{\alpha_m}$, alors $M[P_i/x_i]$ pour $i \in [m]$. Le théorème s'ensuit en choisissant $P_i = x_i$ pour tout $i \in [m]$.

— Si $M = x_1$ est une variable, le résultat est immédiat.
— Si $M = c$ est une constante de type $\tau_1 \to \dots \to \tau_n \to \sigma$ (où σ est un type atomique), alors pour toute séquence de termes réductibles $N_1 \dots N_n$, le terme $c\ N_1 \dots N_n$ de type σ est fortement normalisant, et donc réductible ; donc par définition c est réductible, et le résultat suit.
— Si $M = M_1 M_2$ est une application, alors par hypothèse d'induction, $M_1[P_i/x_i]$ et $M_2[P_i/x_i]$ sont tous deux réductibles. La réductibilité de

l'application $M_1[P_i/x_i]\, M_2[P_i/x_i]$ s'ensuit par définition, puisque ce dernier terme est égal à $M[P_i/x_i]$.

— Si $M = \lambda y.N$ est une abstraction, alors par hypothèse d'induction, $N[P_i/x_i][P/y]$ est réductible pour tout terme P réductible (y étant libre dans N). Par conséquent, le lemme A.2 entraîne que $M[P_i/x_i]$ est réductible.

— Si $M = \phi \longrightarrow M_0$ est une condition, alors par hypothèse d'induction, $M_0[P_i/x_i]$ est réductible, donc $M[P_i/x_i]$ est réductible par **(CR 6)**.

— Si $M = \{M_1 \dots M_n\}$ est un ensemble, alors suivant le même raisonnement, chaque $M_i[P_i/x_i]$ est réductible et la réductibilité de $M[P_i/x_i]$ s'ensuit par **(CR 6)**. □

Annexe B

Propriétés additionnelles du système de réécriture $\rightarrow_{\varepsilon}$

Cette annexe contient les preuves détaillées de plusieurs résultats apparaissant dans dans le chapitre 5, ainsi que plusieurs lemmes intermédiaires utiles à ces démonstrations.

B.1 Résultats de NP-difficulté

Cette section rassemble les démonstrations des résultats de NP-difficulté des différentes classes de grammaires commutatives.

Les deux premières preuves procèdent par réduction du problème 3-PART, similairement à la preuve de difficulté du problème de l'appartenance à un langage commutatif (cf. théorème 5.2). Nous proposons deux grammaires générant le même langage, respectivement une CMREG et une CMG. La NP-difficulté de l'analyse selon une CMCFG fixée s'ensuit immédiatement.

Nous montrons ensuite la NP-difficulté de l'analyse universelle pour les CRG, par réduction du problème NP-complet de 3-couverture exacte d'un ensemble.

B.1.1 Analyse selon une CMREG

Théorème B.1. *L'analyse d'un mot w selon une grammaire régulière multiple commutative $G \in$* **CMREG** *est un problème* NP*-difficile.*

Démonstration. Nous considérons la grammaire commutative G définie par $G = \left(\left\{S^{[0]}, A^{[0,0,0,0]}\right\}, \{a, b, \#\}, S, \mathcal{P}\right)$, avec $\mathcal{P}$ formé par les productions :

1. $S(\otimes[x_1, x_2, x_3, x_4, y]) \leftarrow A(x_1, x_2, x_3, x_4)\ S(y)$
2. $S(\varepsilon) \leftarrow$
3. $A(\odot\ a\ x_1, x_2, x_3, \odot\ b\ x_4) \leftarrow A(x_1, x_2, x_3, x_4)$

4. $A(x_1, \odot\ a\ x_2, x_3, \odot\ b\ x_4) \leftarrow A(x_1, x_2, x_3, x_4)$
5. $A(x_1, x_2, \odot\ a\ x_3, \odot\ b\ x_4) \leftarrow A(x_1, x_2, x_3, x_4)$
6. $A(\#, \#, \#, \#) \leftarrow$

où, par abus de notation, $\otimes[t_1, \ldots, t_n]$ dénote la combinaison libre des termes t_1 à t_n formée par $\otimes\ t_1(\otimes\ t_2\ (\ldots(\otimes\ t_{n-1}\ t_n)\ldots))$.

Considérons maintenant une instance du problème NP-complet 3-PART (défini page 128), constituée d'un ensemble $S = \{n_1 \ldots n_{3m}\}$ et d'un entier k. Nous montrons que le mot $w = a^{n_1}\# \ldots \#a^{n_{3m}}\#(b^k\#)^m$ appartient à $\mathcal{L}(G)$ si et seulement si l'instance correspondante de 3-PART a une solution.

Montrons d'abord la réciproque de l'implication : si l'instance (S, k) de 3-PART a une solution, alors il existe m triplets d'entiers dans S dont la somme vaut k. Pour chacun de ces triplets (x, y, z), considérons la dérivation partant du non-terminal A et utilisant x fois la production 3, y fois la production 4, z fois la production 5 et se concluant par la production 6 (terminale) : le langage de A contient des quadruplets de termes rigides (formés exclusivement par l'opérateur $\odot$) dont les frontières sont $\left(a^x\#, a^y\#, a^z\#, b^k\#\right)$ (puisque $x+y+z = k$) pour chaque triplet (x, y, z) appartenant à la solution de l'instance. Nous combinons ces dérivations en appliquant m fois la production 1 (récursive), et en concluant la dérivation résultante par la production 2. Le terme résultant permet de combiner librement tous les quadruplets précédents, ainsi qu'une occurrence du symbole ε, entraînant l'appartenance de w à son langage (en repoussant le quatrième composant $b^k\#$ de chaque quadruplet à la fin du mot et en réordonnant entre eux les trois premiers termes de chaque).

L'implication directe est ensuite montrée en extrayant de l'analyse de w une solution à l'instance de 3-PART qui lui est associée. Si $w \in \mathcal{L}(G)$, alors il existe deux termes équivalents t et t' tels que $\mathrm{frt}(t') = w$ et $t \in \mathcal{T}(G)$. La structure des productions impose alors (en raison du nombre de symboles # dans w) que t se compose de m fois quatre sous-termes rigides de frontières $a^x\#$, $a^y\#$, $a^z\#$ et $b^{x+y+z}\#$, pouvant être recombinés. Puisque leur réordonnement permet de produire w, un sous-terme de frontière $a^n\#$ est présent pour chaque $n \in S$, et la seconde moitié du mot w impose que chaque facteur $b^{x+y+z}\#$ égale $b^k\#$. La sous-dérivation associée à chaque occurrence du non-terminal A permet alors d'obtenir m triplets formant une partition de S et satisfaisant $x+y+z = k$. □

B.1.2 Analyse selon une CMG

Théorème B.2. *L'analyse d'un mot w selon une grammaire à macros commutative $G \in$* **CMG** *est un problème* NP*-difficile.*

Démonstration. La preuve procède en exploitant une réduction similaire à la précédente, basée sur la grammaire $G = \left(\left\{S^0, A^{0^4 \to 0}\right\}, \{a, b, \#\}, S, \mathcal{P}\right)$, avec $\mathcal{P}$ formé par :

1. $S(\otimes\ (x\ \#\ \#\ \#\ \#)\ y) \leftarrow A(x)\ S(y)$
2. $S(\varepsilon) \leftarrow$
3. $A(\lambda t_1 \dots t_4.x\ (\odot\ a\ t_1)\ t_2\ t_3\ (\odot\ b\ t_4)) \leftarrow A(x)$
4. $A(\lambda t_1 \dots t_4.x\ t_1\ (\odot\ a\ t_2)\ t_3\ (\odot\ b\ t_4)) \leftarrow A(x)$
5. $A(\lambda t_1 \dots t_4.x\ t_1\ t_2\ (\odot\ a\ t_3)\ (\odot\ b\ t_4)) \leftarrow A(x)$
6. $A(\lambda t_1 \dots t_4. \otimes [t_1, t_2, t_3, t_4]) \leftarrow$

Soit (S, k) une instance de 3-PART, nous considérons le même mot associé $w = a^{n_1}\# \dots \#a^{n_{3m}}\#(b^k\#)^m$ que précédemment ; la preuve que l'existence d'une solution équivaut à $w \in \mathcal{L}(G)$ suit également le schéma de celle des CMREG, les deux grammaires reconnaissant le même langage et leurs productions remplissant les mêmes rôles respectifs. □

B.1.3 Analyse universelle selon une CRG

Problème de 3-couverture exacte (X3C) Nous définissons tout d'abord le problème NP-complet X3C :

Instance Une famille $\mathcal{F}$ d'ensembles, telle que chaque ensemble $S \in \mathcal{F}$ se compose de trois nombres entiers n_1, n_2 et n_3 tels que $1 \leq n_i \leq 3m$.

Solution *Vrai* si et seulement si il existe une famille $\mathcal{F}' \subseteq \mathcal{F}$ de m ensembles $S_1 \dots S_m$ couvrant exactement les entiers de 1 à $3m$:

$$\bigcup_{S_i \in \mathcal{F}'} S_i = [3m]$$

Théorème B.3 (Garey et Johnson [1990])**.** *Le problème* X3C *est* NP*-complet.*

Théorème B.4. *L'analyse universelle d'un mot w selon une grammaire commutative régulière $G \in$* **CRG** *est un problème* NP*-complet.*

Démonstration. L'existence d'un algorithme NP résolvant ce problème est une conséquence triviale de la complexité de l'analyse universelle selon une CCFG().

La NP-difficulté est établie par réduction à X3C. Nous construisons une CRG G et un mot w à partir d'une famille $\mathcal{F}$ constituant une instance de X3C, tels que $w \in \mathcal{L}(G) \Leftrightarrow \mathbf{X3C}(\mathcal{F}) = \mathit{Vrai}$. Soit l'alphabet $\Sigma = [3m]$ formé par l'ensemble des entiers de 1 à $3m$ inclus ; nous construisons $w = 1 \dots 3m$ et $G = (\{A\}, \Sigma, A, \mathcal{P})$, où $\mathcal{P}$ permet de réécrire A soit en ε, soit en la combinaison

libre de A avec les trois entiers d'un des ensembles S de $\mathcal{F}$, comme suit :

$$\mathcal{P} = \left\{ A\left(\begin{array}{ccccc} & & \otimes & & \\ & \swarrow & & \searrow & \\ n_1 & & & & \otimes \\ & & & \swarrow & \\ & & n_2 & & \otimes \\ & & & \swarrow & \\ & & & n_3 & x \end{array} \right) \leftarrow A(x) \mid \{n_1, n_2, n_3\} \in \mathcal{F} \right\} \cup \{A(\varepsilon) \leftarrow\}$$

Nous montrons d'abord que l'existence d'une solution à l'instance $\mathcal{F}$ de X3C entraîne que $w \in \mathcal{L}(G)$. Si $\mathcal{F}$ a une solution, alors il existe un une famille $\mathcal{F}' = \{S_1 \dots S_m\}$ dont l'union est égale à l'ensemble $[3m]$. Dans ce cas, nous pouvons successivement réécrire l'axiome A de G à l'aide des productions associées aux ensembles S_i de $\mathcal{F}'$, puis en ε. Le terme résultant appartient au langage de termes de G et combine à l'aide de l'opérateur $\otimes$ une et une seule occurrence de chaque entier appartenant à $[3m]$. Par conséquent, l'associativité et la commutativité de $\otimes$ nous permettent de construire un terme équivalent dont la frontière est exactement w, montrant que $w \in \mathcal{L}(G)$.

Nous montrons maintenant l'implication inverse. Supposons que $w \in \mathcal{L}(G)$; il existe alors un terme t' de frontière w équivalent à un terme $t \in \mathcal{T}(G)$. Il existe donc une dérivation selon G utilisant m productions non-terminales (chaque production ajoutant trois lettres à la frontière de t). Ces m productions sont deux à deux distinctes (puisque w ne contient pas deux fois la même lettre) et correspondent chacune à un des triplets de $\mathcal{F}$. L'union des lettres qu'elles ajoutent à t est exactement l'ensemble $\Sigma = [3m]$; la famille $\mathcal{F}'$ formée à partir des ensembles de $\mathcal{F}$ correspondant aux productions utilisées dans la dérivation de t est donc une solution à l'instance $\mathcal{F}$ de X3C. □

B.2 Algorithmes d'analyse polynomiaux

Cette section contient les preuves de correction, complétude et complexité des algorithmes d'analyse à grammaire fixée détaillés dans la section 5.5.

B.2.1 Reconnaissance selon une CRG

Théorème B.5. *Étant donné une* CRG G *et un mot* $w = l_1 \dots l_n$ *de longueur* $|w| = n$*, l'algorithme d'analyse rappelé dans la figure B.1 permet de dériver* $\top$ *à partir du fait* $(S, \vec{0}, S, 0, n)$ *si et seulement si* $w \in \mathcal{L}(G)$*, et peut être exécuté en espace logarithmique par rapport à* n*.*

Démonstration. La preuve de correction et complétude de l'algorithme procède en montrant que celui-ci dérive un fait de la forme (A, v, B, i, j) (avec $\|v\| \leq n$) si et seulement si les affirmations suivantes sont vraies :

1. le langage étendu de S contient un terme $C[A]$,
2. le terme $C[A]$ équivaut à un terme t tel que $\mathrm{frt}(t) = l_1 \dots l_i\, A\, l_j \dots l_n$,
3. le langage commutatif de A contient un terme $C'[B]$,
4. le terme $C'[B]$ est tel que $\psi(\mathrm{frt}(C'[B])) = v + \vec{1}_B$.

Nous montrons d'abord la correction de l'algorithme, en établissant l'implication directe de l'affirmation précédente par induction sur la dérivation d'un fait composite. Considérons chacune des trois règles susceptibles de conclure cette dérivation :

ANALYSE Dans ce cas, le fait résultant est de la forme $(B, \vec{0}, B, i + |a|, j)$; en appliquant l'hypothèse d'induction au fait précédent $(A, \vec{0}, A, i, j)$, nous déduisons que le langage étendu de S contient un terme $C_{HI}[A]$, qui est équivalent à un terme de frontière $l_1 \dots l_i\, A\, l_j \dots l_n$ (par les affirmations 1 et 2). Par suite, la seconde pré-condition de la règle ANALYSE entraîne que $C_{HI}[\odot\ a\ B]$ appartient à son tour au langage étendu de S ; il équivaut en outre à un terme de frontière $l_1 \dots l_i\, a\, B\, l_j \dots l_n$. Nous pouvons donc choisir $C = C_{HI}[\odot\ a\ x]$ et le contexte trivial $C' = x$ et vérifier que les affirmations 1 à 4 correspondant au fait résultant $(B, \vec{0}, B, i + |a|, j)$ sont vraies (en utilisant la dernière condition de la règle pour établir que a est soit ε, soit l_{i+1}).

HYPOTHÈSE Dans ce cas, le fait résultant est de la forme $(A, v + \vec{1}_a, C, i, j)$; nous appliquons une fois encore l'hypothèse d'induction au fait précédent (A, v, B, i, j), en établissant cette fois que le langage commutatif de A contient un terme $C'_{HI}[B]$ dont la frontière a pour image de Parikh $v + \vec{1}_B$ (par les affirmations 3 et 4). La seconde pré-condition de la règle nous permet alors d'établir que le langage commutatif de A contient également $C'_{HI}[\otimes\ a\ C]$ dont la frontière forme le vecteur $v + \vec{1}_a + \vec{1}_C$. Les deux premières affirmations de l'hypothèse d'induction nous fournissent en outre un terme $C_{HI}[A]$ appartenant au langage étendu de S, équivalent à un terme de frontière $l_1 \dots l_i\, A\, l_j \dots l_n$. Nous posons alors les contextes $C = C_{HI}$ et $C' = C'_{HI}[\otimes\ a\ x]$ pour vérifier les affirmations 1 à 4 de $(A, v + \vec{1}_a, C, i, j)$, et concluons.

CONFIRMATION Dans ce dernier cas, le fait résultant est $(B, \vec{0}, B, i', j')$, dérivé à partir du fait (A, v, B, i, j) auquel nous appliquons l'hypothèse d'induction. Celle-ci produit deux contextes C_{HI} et C'_{HI} vérifiant les affirmations 1 à 4 comme dans le cas précédent, de sorte que $C_{HI}[x]$ équivaut à un contexte de frontière $l_1 \dots l_i\, x\, l_j \dots l_n$ et $C'_{HI}[x]$ équivaut à n'importe quel contexte de frontière $v + \vec{1}_x$ (par définition du langage commutatif d'un non-terminal) ; or $v = \psi(w, i, i') + \psi(w, j', j)$ par la dernière pré-condition de la règle CONFIRMATION. Considérons maintenant le contexte $C = C_{HI}[C'_{HI}]$: le terme $C[B]$ appartient au langage étendu de S (par les affirmations 1 et 3), et équivaut à un terme de frontière $l_1 \dots l_i \dots l_{i'}\, B\, l_{j'} \dots l_j \dots l_n$ (en combinant ce que nous savons de

C_{HI} et C'_{HI}), vérifiant les affirmations 1 et 2 associées au fait résultant $(B, \vec{0}, B, i', j')$; les affirmations 3 et 4 étant satisfaites par le contexte trivial $C' = x$ de frontière vide, concluant l'induction.

Nous vérifions finalement la correction de l'algorithme en examinant la règle RÉUSSITE qui conclut toute dérivation de $\top$: puisque le fait (A, v, B, i, j) est dérivable, les affirmations 1 à 4 sont vraies d'après l'induction précédente, et $v + \vec{1}_a = \psi(w, i, j)$. Nous combinons une fois encore les contextes C et C' pour former un terme $C[C'[a]]$: celui-ci appartient au langage de l'axiome S et équivaut à un terme de frontière $l_1 \dots l_n = w$, établissant finalement que $w \in \mathcal{L}(G)$.

Nous montrons maintenant l'implication réciproque, et en déduisons la complétude de l'algorithme. Nous procédons par induction sur la taille des deux contextes C et C' fournis par les affirmations 1 à 4. Dans le cas de base (s'ils sont tous les deux triviaux), $l_1 \dots l_i = l_j \dots l_n = \varepsilon$, et $v = \vec{0}$, par conséquent le fait correspondant est $(S, \vec{0}, S, 0, n)$, qui est l'axiome de départ de l'algorithme.

Dans le cas contraire, nous décroissons d'abord sur la taille du contexte C', que nous supposons non-trivial. Nous savons par l'affirmation 3 que $C'[B]$ appartient au langage commutatif de A : par conséquent il existe un contexte C'' tel que $C' = C''[\otimes\ a\ x]$, et un non-terminal N qui se réécrit en $\otimes\ a\ B$; en outre, $C''[N]$ appartient également au langage commutatif de A. Reprenant les affirmations 1 et 2, et en considérant les affirmations 3 et 4 appliquées au contexte C'' (dont la frontière produit le vecteur $v - \vec{1}_a$), nous pouvons appliquer notre hypothèse d'induction pour établir que le fait $(A, v - \vec{1}_a, N, i, j)$ est dérivable. Or, nous venons d'établir que $N(\otimes\ a\ x) \leftarrow B(x)$ est une production de $\mathcal{P}$, et nous savons que $\|v - \vec{1}_a\| \leq n$. La règle HYPOTHÈSE nous permet donc d'établir le fait (A, v, B, i, j), montrant l'implication réciproque dans ce cas.

Supposons maintenant que C' est trivial ; nous décroissons alors sur la taille du contexte C. Nous distinguerons deux cas, que nous ferons correspondre respectivement aux règles ANALYSE et CONFIRMATION, en fonction du dernier opérateur apparaissant dans C ($\odot$ ou $\otimes$). Dans tous les cas nous savons que, d'une part, le contexte C' est trivial, et par conséquent $v = \vec{0}$ et B appartient au langage commutatif de A : en l'absence de règles unitaires (G étant en forme normale), nous pouvons en déduire que $A = B$; d'autre part, le terme $C[A]$ appartient au langage étendu de S. Nous considérons maintenant deux cas, en fonction du dernier opérateur présent dans C :

— Supposons d'abord que $C[A]$ est de la forme $C''[\odot\ a\ A]$. Il existe alors un non-terminal N se réécrivant en $\odot\ a\ A$, et le terme $C''[N]$ équivaut (en utilisant l'affirmation 1) à un terme de frontière $l_1 \dots l_{i-|a|}\, N\, l_j \dots l_n$, avec, suivant le cas, $a = \varepsilon$ ou $a = l_i$. Nous appliquons l'hypothèse d'induction aux affirmations 1 à 4 en y substituant C'' à C et N à A et B, établissant ainsi que le fait $(N, \vec{0}, N, i - |a|, j)$ est dérivable. La réécri-

ture possible de N en $\odot$ a A et la valeur de a établies précédemment nous permettent alors d'appliquer la règle ANALYSE pour dériver le fait $(A, \vec{0}, A, i, j)$, vérifiant l'induction (puisque $A = B$ et $v = \vec{0}$).

— Supposons maintenant que le dernier opérateur dans C est $\otimes$. Dans ce cas, nous posons $C[A] = R[L[A]]$, où le contexte R à la racine de C contient le début et la fin de w, et le contexte L, qui utilise exclusivement l'opérateur $\otimes$, permet de réordonner librement sa frontière. Nous choisissons également un non-terminal N apparaissant dans la dérivation de $C[A]$ qui permette d'établir les affirmations suivantes :

1. $R[N]$ appartient au langage étendu de S
2. $R[N]$ équivaut à un terme de frontière $l_1 \dots l_{i'} N l_{j'} \dots l_n$
3. le langage commutatif de N contient le terme $L[B]$

Selon la structure du terme de $\mathcal{T}(G)$ qui reconnaît w, plusieurs N sont susceptibles de vérifier ces affirmations. Il en existe cependant toujours au moins un : le non-terminal qui domine le plus grand contexte L contenant uniquement l'opérateur $\otimes$ établit toujours ces affirmations, car aucune des lettres de R (qui se conclut alors par $\odot$ ou est trivial) n'est dans ce cas susceptible d'échanger sa position avec une des lettres de L pour former w (la figure 5.13 permet de mieux visualiser ce fait). Nous posons alors $\psi(\mathrm{frt}(L[B])) = u + \vec{1}_B$ et appliquons l'hypothèse d'induction aux affirmation établies par N (puisque R est plus petit que C), et concluons que le fait (N, u, B, i', j') est dérivable. De plus, le terme $L[B]$ appartient au langage commutatif de N, et le vecteur u contient exactement les lettres de L (présentes dans C mais absentes de R), à savoir $l_{i'+1} \dots l_i$ et $l_j \dots l_{j'-1}$. Par conséquent, nous pouvons appliquer la règle CONFIRMATION pour dériver le fait $(B, \vec{0}, B, i, j)$, concluant l'induction puisque $A = B$ et $v = \vec{0}$.

Comme pour la preuve de correction, nous concluons la preuve en considérant la pénultième dérivation d'un terme t équivalent à un terme de frontière w. Celle-ci fournit deux contextes C et C' permettant de dériver, par l'induction précédente, un fait (A, v, B, i, j). Cette dérivation est complétée par l'emploi d'une production terminale $B(a)$, de sorte que $v + \vec{1}_a = \psi(l_i \dots l_j)$, et nous pouvons finalement appliquer la règle RÉUSSITE pour dériver le fait $\top$, montrant la complétude de l'algorithme.

Enfin, nous prouvons la complexité de l'algorithme, en montrant d'abord que tout fait de dérivation peut être représenté en utilisant au plus $\mathcal{O}(\log(n))$ en mémoire. C'est immédiatement le cas pour les non-terminaux A et B (de taille constante), et les indices i et j (puisque $1 \leq i, j \leq n$). Par ailleurs la règle HYPOTHÈSE est la seule qui permet de faire croître v et impose que $\|v\| \leq n$; la dimension de v étant constante (puisque Σ et $\mathcal{N}$ sont fixés), il est également représentable en espace logarithmique par rapport à n.

Par suite, il est immédiat que toute règle peut être individuellement ap-

pliquée en respectant les contraintes de complexité de l'algorithme ; en effet, l'ensemble $\psi(G, A)$ étant semilinéaire, l'appartenance de $v + \vec{1}_B$ à cet ensemble peut être testée en espace logarithmique par rapport à $\|v\|$ (comme mentionné section 2.1.4). Les règles de l'algorithme étant séquentielles, il en résulte la complexité attendue. □

$$\frac{(A, \vec{0}, A, i, j) \quad A(\odot\ a\ x) \leftarrow B(x) \in \mathcal{P} \quad a = l_{i+1} \vee a = \varepsilon}{(B, \vec{0}, B, i + |a|, j)}\ \textsc{Analyse}$$

$$\frac{(A, v, B, i, j) \quad B(\otimes\ a\ x) \leftarrow C(x) \in \mathcal{P} \quad \|v\| \leq |w|}{(A, v + \vec{1}_a, C, i, j)}\ \textsc{Hypothèse}$$

$$\frac{(A, v, B, i, j) \quad v + \vec{1}_B \in \psi(G, A) \quad v = \psi(w, i, i') + \psi(w, j', j)}{(B, \vec{0}, B, i', j')}\ \textsc{Confirmation}$$

$$\frac{(A, v, B, i, j) \quad B(a) \leftarrow \in \mathcal{P} \quad v + \vec{1}_a = \psi(w, i, j)}{\top}\ \textsc{Réussite}$$

FIGURE B.1 – Algorithme d'analyse selon une CRG

Observons finalement que toutes les composantes $v.A$ (où $A \in \mathcal{N}$) de v sont nulles durant le fonctionnement de l'algorithme : ces composantes n'entrent en jeu que lors de l'application de la règle CONFIRMATION, qui y ajoute un vecteur unitaire $\vec{1}_B$). Les vecteurs manipulés peuvent donc être considéré comme des éléments de $\mathbb{N}^{\Sigma}$, sans tenir compte de $\mathcal{N}$. Par conséquent, la complexité en espace de l'algorithme est également logarithmique par rapport à G si la taille de l'alphabet Σ est bornée par une constante. L'analyse universelle selon une CRG, qui constitue dans le cas général un problème NP-complet, peut donc être effectuée en temps polynomial si l'alphabet support de la grammaire est fixé.

B.2.2 Reconnaissance selon une CCFG

Théorème B.6. *Étant donné une* CCFG G *et un mot* $w = l_1 \dots l_n$*, l'algorithme d'analyse rappelé dans la figure B.2 permet de dériver le fait* $(\vec{1}_S, 0, n)$ *si et seulement si* $w \in \mathcal{L}(G)$*, et appartient à la classe de complexité* **LOGCFL**.

Démonstration. Comme pour la preuve de l'algorithme d'analyse des CRG, nous montrons inductivement l'équivalence entre la dérivabilité d'un fait et un ensemble d'affirmations portant sur les langages des non-terminaux de G, et en déduisons la correction et la complétude de l'algorithme. Ainsi, nous montrons qu'un fait (v, i, j) est dérivable si et seulement si ces affirmations sont vraies :

1. il existe $w' = A_1 \dots A_m$ tel que $\psi(w') = v$,

2. $l_{i+1} \dots l_j = w_1 \dots w_m$ tel que pour tout $i \in [m]$, $w_i \in \mathcal{L}(A_i)$.

Par abus de notation, le facteur $l_{i+1} \dots l_j$ de w désigne le mot vide si $i \geq j$. Le mot w' en revanche, est supposé non-vide dans tous les cas (aucun fait de la forme $(\vec{0}, i, j)$ n'est dérivable).

Nous prouvons d'abord la correction de l'algorithme, en montrant l'implication directe dans l'équivalence précédente. Nous procédons par induction sur la dérivation de (v, i, j), en considérant chacune des règles susceptibles de conclure cette dernière :

VIDE Dans ce cas, $i = j$ et v est un vecteur unitaire $\vec{1}_A$. Nous posons donc $w' = A$, satisfaisant immédiatement les affirmations 1 et 2 car $\varepsilon \in \mathcal{L}(A)$.

CONSTANTE Dans ce cas, $i = j - 1$ et $v = \vec{1}_A$. Nous posons donc à nouveau $w' = A$, satisfaisant immédiatement les affirmations 1 et 2 car $l_i \in \mathcal{L}(A)$.

ANALYSE Dans ce cas, $v = \vec{1}_A$ et, en raison des pré-conditions de la règle, deux faits $(\vec{1}_B, i, k)$ et $(\vec{1}_C, k, j)$ sont dérivables. En appliquant l'hypothèse d'induction aux deux faits précédents, nous obtenons que $l_{i+1} \dots l_k$ appartient à $\mathcal{L}(B)$ et $l_{k+1} \dots l_j$ à $\mathcal{L}(C)$ (puisque, suivant le cas, $w' = B$ ou $w' = C$ respectivement). Or A se réécrit en $\odot\ B\ C$ en raison de la troisième pré-condition, donc si $w' = A$ (satisfaisant l'affirmation 1), le mot $l_{i+1} \dots l_k\, l_{k+1} \dots l_j$, concaténant les deux facteurs précédents, appartient effectivement à $\mathcal{L}(A)$, satisfaisant l'affirmation 2.

RÉDUCTION Dans ce cas, nous savons qu'un fait (v, i, j) est dérivable, avec $v \in \psi(G, A)$. Par hypothèse d'induction sur (v, i, j), il existe un mot w" formé par les symboles non-terminaux de v, et appartenant donc au langage commutatif de A. De plus, les symboles de w" se réécrivent en une série de facteurs distincts formant le mot $l_{i+1} \dots l_j$ (affirmation 2 de l'hypothèse d'induction). Par suite, $l_{i+1} \dots l_j$ appartient à $\mathcal{L}(A)$ (celui-ci incluant, par le langage commutatif de A, un terme équivalent à un terme de frontière w"). Nous posons donc $w' = A$, vérifiant les affirmations 1 et 2.

COMBINAISON Dans ce cas, le vecteur v vérifie $v = u_1 + u_2$, et il existe un indice k tel que les faits (u_1, i, k) et (u_2, k, j) sont dérivables. Aussi, nous pouvons établir les affirmations suivantes en utilisant à deux reprises l'hypothèse d'induction :

— il existe un mot $b = B_1 \dots B_l$ tel que $\psi(b) = u_1$,
— $l_{i+1} \dots l_k = b_1 \dots b_l$ où chaque b_i appartient à $\mathcal{L}(B_i)$;
— il existe un mot $c = C_1 \dots C_p$ tel que $\psi(c) = u_2$
— $l_{k+1} \dots l_j = c_1 \dots c_p$ où chaque c_i appartient à $\mathcal{L}(C_i)$;

Par suite, nous posons $m = l + p$, et A_i et w_i désignent soit B_i et b_i (lorsque $i \leq l$), soit C_{i-l} et c_{i-l} (lorsque $i > l$). Il s'ensuit immédiatement qu'il existe $w' = A_1 \dots A_m$ ayant pour image de Parikh $u_1 + u_2 = v$ (affirmation 1), et que $l_{i+1} \dots l_k\, l_{k+1} \dots l_j = w_1 \dots w_m$ avec $w_i \in \mathcal{L}(A_i)$, vérifiant l'affirmation 2 et concluant l'induction.

Par suite, si le fait $(\vec{1}_S, 0, n)$ est dérivable, alors le mot $w = l_1 \dots l_n$ appartient

à $\mathcal{L}(S)$, et donc $w \in \mathcal{L}(G)$.

Nous montrons maintenant la réciproque de l'implication, afin d'établir la complétude de l'algorithme. Nous supposons la validité des affirmations 1 et 2, et montrons qu'un fait (v, i, j) correspondant est dérivable, par induction sur la somme des tailles des dérivations montrant que $w_i \in \mathcal{L}(A_i)$. En raison de l'affirmation 1, il existe un mot $w' = A_1 \dots A_m$; nous distinguons deux cas selon la longueur de w' : soit $m > 1$, soit $m = 1$ (par hypothèse, $w' \neq \varepsilon$).

Dans le premier cas, le vecteur v correspondant n'est pas unitaire, et la règle COMBINAISON doit donc être appliquée pour dériver (v, i, j). Nous choisissons $0 < p < m$ et en tirons les quatre affirmations suivantes :

3. il existe $w_g = A_1 \dots A_p$ tel que $\psi(w_g) = u_1$,
4. $l_{i+1} \dots l_k = w_1 \dots w_p$ tel que pour tout $i \in [p]$, $w_i \in \mathcal{L}(A_i)$,
5. il existe $w_d = A_{p+1} \dots A_m$ tel que $\psi(w_d) = u_2$,
6. $l_{k+1} \dots l_j = w_{p+1} \dots w_m$ tel que pour tout $p < i \leq m$, $w_i \in \mathcal{L}(A_i)$.

La somme des tailles des dérivations impliquées dans les affirmations 4 et 6 (montrant que $w_i \in \mathcal{L}(A_i)$) est immédiatement strictement inférieure à celle de l'affirmation 2. Par conséquent, nous pouvons appliquer l'hypothèse d'induction sur les affirmations 3,4 et 5,6 pour établir que les faits (u_1, i, k) et (u_2, k, j) sont dérivables. Or $v = u_1 + u_2$, donc la règle COMBINAISON permet de dériver le fait (v, i, j), nous permettant de conclure dans le cas où $m > 1$.

Nous supposons maintenant que $m = 1$. Nous notons $w' = A_1 = A$, $v = \vec{1}_A$ et t est le terme de $\mathcal{T}(A)$ permettant d'établir que $w_1 = l_{i+1} \dots l_j$ appartient à $\mathcal{L}(A)$. Nous distinguons quatre cas de figure en fonction de la structure de t et de sa dérivation (observons que l'induction procède sur la taille de celle-ci, puisque l'affirmation 2 ne suppose qu'une seule dérivation lorsque $m = 1$).

— Supposons d'abord que $\mathrm{frt}(t) = \varepsilon$. Ce cas est résolu immédiatement puisqu'il entraîne que $l_{i+1} \dots l_j = \varepsilon$ et donc que $i = j$. Le fait à dériver est donc (A, i, i), est peut-être immédiatement obtenu en appliquant la règle VIDE, le terme t satisfaisant son unique pré-condition.
Une conséquence particulière est que nous pouvons supposer à l'avenir que $t \neq \varepsilon$; nous considérons donc les trois cas restants : soit $t = a$, soit $t = \odot\ t_1\ t_2$, soit enfin $t = \otimes\ t_1\ t_2$.

— Si $t = a$, alors $l_{i+1} \dots l_j = a$; par conséquent, $i = j - 1$ et $a = l_j$. Puisque G est en forme normale, le terme t doit avoir été dérivé par une production $A(a) \leftarrow$; nous pouvons donc appliquer la règle CONSTANTE pour dériver le fait $(\vec{1}_A, j - 1, j)$ attendu.

— Si $t = \odot\ t_1\ t_2$, alors G contient une production de la forme $A(\odot\ x\ y) \leftarrow B(x)\ C(y)$. En outre, nous pouvons séparer le mot $l_{i+1} \dots l_k\, l_{k+1} \dots l_j$ en deux composantes, à une position k. La première vérifie les deux affirmations suivantes :
  — il existe $w" = A_1 \dots A_p$ tel que $\psi(w_g) = \vec{1}_B$
  — $l_{i+1} \dots l_k = w_1 \dots w_p$ tel que pour tout $i \in [p]$, $w_i \in \mathcal{L}(A_i)$

Ce qui entraîne, par hypothèse d'induction, que le fait $(\vec{1}_B, i, k)$ est dérivable. Suivant le même raisonnement, la seconde composante permet d'établir la dérivabilité du fait $(\vec{1}_C, k, j)$. Observons que cette séparation est possible en raison de la présence de l'opérateur $\odot$ à la racine de t : celle-ci entraîne que tout terme équivalent à t a pour frontière la concaténation de deux mots distincts (la frontière d'un terme équivalent à t_1 et celle d'un terme équivalent à t_2). Pour conclure, la production de G qui permet de construire t et les deux faits établis comme dérivables en utilisant l'hypothèse d'induction permettent, en utilisant la règle ANALYSE, de dériver le fait $(\vec{1}_A, i, j)$.

— Enfin, nous traitons le cas où $t = \otimes\ t_1\ t_2$. Nous considérons alors un vecteur u, défini comme l'image de Parikh associée à la plus grande sous-dérivation de t qui ne contienne que des symboles non-terminaux et l'opérateur $\otimes$. En d'autres termes, cette dérivation est obtenue en considérant celle de t et en excluant toutes les étapes de réécritures requérant une production terminale ou l'emploi de l'opérateur $\odot$: la frontière du terme obtenu se compose exclusivement de symboles non-terminaux, et son image de Parikh u appartient par définition au langage commutatif $\psi(G, A)$ de A. Nous en tirons les deux affirmations suivantes :
  — il existe $w" = A_1 \dots A_m$ tel que $\psi(w") = u$
  — $l_{i+1} \dots l_j = w_1 \dots w_m$, avec $w_i \in \mathcal{L}(A_i)$ pour tout $i \in [m]$
  Le mot $w"$ réordonne simplement les symboles non-terminaux formant la frontière de la sous-dérivation décrite plus haut. L'appartenance des mots w_i à $\mathcal{L}(A_i)$ est donnée par les sous-dérivations précédemment exclues (utilisant des productions terminales ou l'opérateur $\odot$), dont la somme des tailles est strictement inférieure à celle de la dérivation originale (au moins une occurrence de l'opérateur $\otimes$, à la racine de t, en est exclue, requérant un pas de réécriture à partir de A). Par conséquent, notre hypothèse d'induction montre que le fait (u, i, j) est dérivable et, puisque $u \in \psi(G, A)$, la règle RÉDUCTION nous permet de dériver le fait $(\vec{1}_A, i, j)$, concluant la preuve de l'implication réciproque.

Finalement, si un mot w est dérivable selon G, alors il existe $w' = S$ (de vecteur associé $\vec{1}_S$), tel que $w = l_1 \dots l_n$ appartient à $\mathcal{L}(S)$. Par conséquent, comme nous venons de l'établir, le fait $(\vec{1}_S, 0, n)$ est dérivable, ce qui montre la complétude de l'algorithme.

Nous montrons maintenant que cet algorithme appartient à la classe de complexité LOGCFL, définie comme l'ensemble des problèmes réductibles en espace logarithmique à l'appartenance d'un mot à un langage hors-contexte. Nous exploitons pour ce faire une caractérisation de cette classe établie par Ruzzo [1980], qui correspond à l'ensemble des problèmes décidables par une machine de Turing alternante[Chandra *et al.*, 1981] en espace logarithmique et avec un arbre de calcul de taille polynomiale. Sans donner de construction

précise, nous expliquons le fonctionnement général d'une machine de Turing alternante qui implémente l'algorithme, et donnons les arguments clés permettant d'établir que celle-ci respecte les contraintes évoquées ci-dessus en termes de mémoire et de taille de l'arbre d'exécution.

Considérons le système de règles donné par la figure B.1 : chacune de ses règles dérive un fait de la forme (v, i, j), où $0 \leq i, j \leq n$ et v est un vecteur de dimension constante $(\mathcal{N})$ et dont la norme supremum est bornée par $n + 1$. Par conséquent, ces faits peuvent être représentés en espace logarithmique par rapport à $|w| = n$. La machine de Turing alternante qui implémente cet algorithme choisit une règle de l'algorithme (ainsi qu'une valeur pour ses prémisses) lorsqu'elle est dans un état existentiel, et prouve ensuite chacune des prémisses de cette règle depuis un état universel. Cette machine représente originellement sur sa bande le fait $(\vec{1}_S, 0, n)$, et accepte le mot w passé en entrée si et seulement si elle parvient à éliminer toutes les prémisses des règles qu'elle choisit d'appliquer.

Puisque la bande de la machine ne contient à tout instant qu'un unique fait dépendant de $|w|$, celle-ci fonctionne en espace logarithmique par rapport à n ; reste à borner polynomialement la taille de l'arbre de calcul. Celui-ci peut être vu comme le nombre d'états effectivement traversés lors d'un calcul réussi minimal de la machine : il inclut à chaque transition depuis un état existentiel la taille minimale d'un arbre de calcul réussi parmi les successeurs possibles, et à chaque transition depuis un état universel la somme des tailles minimales des arbres de calculs des différents successeurs. La borne que nous établissons repose sur deux observations clés : d'une part, il n'existe qu'un nombre polynomial de faits dérivables lors de l'analyse d'un mot w ; d'autre part, il existe une métrique sur ces faits qui décroît rapidement lors de toute transition depuis un état universel.

Observons tout d'abord que toutes les prémisses des règles de l'algorithme sont soit des faits de dérivation, soit vérifiables dans NL (dans le cas de la règle Réduction), soit en temps constant (pour toutes les autres règles). Il suffit donc de borner le nombres d'étapes de calcul servant à établir des faits de dérivation. Compte tenu de la borne sur $\|v\|$ et des valeurs possibles de i et j, le nombre de faits de dérivation distincts est borné par n^{N+2}, où $N = \#(\mathcal{N})$ est le nombre (constant) de symboles non-terminaux dans G. Par conséquent, toute branche de calcul est de hauteur au plus polynomiale : en effet, si une branche de calcul possède une hauteur supérieure n^{N+2}, alors elle cherche à établir au moins deux fois un même fait et peut donc être élaguée. Nous considérons maintenant la *taille* d'une étape de calcul, définie comme la taille $j - i$ du fragment de w qu'elle reconnaît : tout arbre de calcul comprenant une étape de taille 1 ou 0 se conclut immédiatement par une application de la règle Constante ou Vide. En outre, toute étape de taille supérieure à 1 possède soit un seul successeur par une transition universelle (règle Réduction), soit deux successeurs tels que la somme de leurs tailles égale celle l'étape précé-

dente (règles Analyse et Combinaison). Aussi, puisque la hauteur de l'arbre de calcul est bornée polynomialement, et que sa largeur est bornée logarithmiquement à terme, la taille totale de l'arbre demeure bornée polynomialement, entraînant que le problème de l'analyse d'un mot selon une CCFG appartient à LogCFL. □

$$\frac{A(l_i) \leftarrow \in \mathcal{P}}{(\vec{1}_A, i-1, i)}\ \textsc{Constante} \qquad \frac{\exists t \in \mathcal{T}_G(A).\, \mathrm{frt}(t) = \varepsilon}{(\vec{1}_A, i, i)}\ \textsc{Vide}$$

$$\frac{(\vec{1}_B, i, j) \quad (\vec{1}_C, j, k) \quad A(\odot\ x\ y) \leftarrow B(x)\ C(y) \in \mathcal{P}}{(\vec{1}_A, i, k)}\ \textsc{Analyse}$$

$$\frac{(u, i, j) \quad (v, j, k) \quad \|u+v\| \leq |w| + 1}{(u+v, i, k)}\ \textsc{Combinaison}$$

$$\frac{(v, i, j) \quad v \in \psi(G, A)}{(\vec{1}_A, i, j)}\ \textsc{Réduction}$$

Figure B.2 – Algorithme d'analyse selon une CCFG

B.3 Algorithme d'analyse général

Cette section contient les démonstrations des différentes propriétés du système de typage sémantique des termes proposé dans la section 5.6. Elle se conclut par les preuves de correction, complétude et complexité de l'algorithme de reconnaissance général présenté à l'issue du chapitre 5.

B.3.1 Système de typage sémantique et β-réduction

Nous rappelons tout d'abord dans la figure B.3 les règles du système de typage sémantique que nous étudions. Nous démontrons ensuite les propriétés listées dans le chapitre 5, portant sur les dérivations engendrées par ce système et leur compatibilité avec la β-réduction dans le cas des termes affines.

Lemme B.7 (Renforcement)**.** *Si* $\Gamma, x : \mathbf{a} \vdash M : \mathbf{b}$ *est dérivable et que* x *n'est pas libre dans* M*, alors* $\Gamma \vdash M : \mathbf{b}$ *est dérivable.*

Démonstration. La preuve procède par induction sur la dérivation $\mathtt{M}$ du jugement de départ, construisant une dérivation $\mathtt{M}_+$ du second jugement :

cst Dans ce cas, le second jugement est immédiatement dérivable par *cst*.

var Dans ce cas, $M \neq x$ (puisque par hypothèse, $x \notin \mathrm{FV}(M)$), et le résultat est également immédiat.

$$\frac{}{\Gamma, x : \mathbf{a} \vdash x : \mathbf{a}}\, var \qquad \frac{c \in C \quad \mathbf{a} \in \tau(c)}{\Gamma \vdash M : \mathbf{a}}\, cst$$

$$\frac{\Gamma, x : \mathbf{a} \vdash M : \mathbf{b}}{\Gamma \vdash \lambda x.M : \mathbf{a} \to \mathbf{b}}\, abs \qquad \frac{\Gamma \vdash M : \mathbf{a} \to \mathbf{b} \quad \Delta \vdash N : \mathbf{a}}{\Gamma \vdash MN : \mathbf{b}}\, app$$

$$\frac{\Gamma \vdash M : \mathbf{b} \quad x \notin \Gamma}{\Gamma \vdash \lambda x.M : V_\alpha \to \mathbf{b}}\, abs, \bot \qquad \frac{\Gamma \vdash M : V_\alpha \to \mathbf{b}}{\Gamma \vdash MN^\alpha : \mathbf{b}}\, app, \bot$$

FIGURE B.3 – Rappel des règles de dérivation des types sémantiques

abs/**abs**, $\bot$/**app**, $\bot$ Dans ces cas, le résultat s'ensuit en appliquant l'hypothèse d'induction à la sous-dérivation immédiate de $\mathtt{M}$, puis en ajoutant à la dérivation résultante la dernière règle de $\mathtt{M}$ pour obtenir $\mathtt{M}_+$.

app Dans ce cas, $\mathtt{M}$ combine deux dérivations, lesquelles dérivent deux jugements dont les environnements de typages sont disjoints. Par conséquent, $x : \mathbf{a}$ doit appartenir à un seul de ces environnements : nous appliquons l'hypothèse d'induction à la dérivation correspondante, et combinons le résultat avec l'autre dérivation par *app* pour obtenir $\mathtt{M}_+$. □

Lemme B.8 (Affaiblissement)**.** *Si* $\Gamma \vdash M : \mathbf{b}$ *est dérivable et que* $\Gamma, \Delta \vdash M : \mathbf{b}$ *est bien formé, alors* $\Gamma, \Delta \vdash M : \mathbf{b}$ *est dérivable.*

Démonstration. Une fois encore encore la preuve procède par induction sur la dérivation d'origine. Dans tous les cas, le résultat s'ensuit immédiatement en appliquant l'hypothèse d'induction là où c'est possible, à l'exception de la règle *app* : dans ce cas le résultat s'ensuit en appliquant l'hypothèse d'induction à l'une ou l'autre des sous-dérivations et en combinant avec l'autre sous-dérivation le résultat obtenu par *app*. □

Lemme B.9 (Lemme de substitution)**.** *Ces deux affirmations sont vraies :*

1. *S'il existe deux dérivations* $\mathtt{M} :: \Gamma, x : \mathbf{a} \vdash M : \mathbf{b}$ *et* $\mathtt{N} :: \Delta \vdash N : \mathbf{a}$*, alors* $\Gamma, \Delta \vdash M[N/x] : \mathbf{b}$ *est dérivable.*
2. *S'il existe une dérivation* $\mathtt{M} :: \Gamma \vdash M : \mathbf{b}$ *et que* x^α *n'est pas dans* Γ*, alors* $\Gamma \vdash M[N^\alpha/x^\alpha] : \mathbf{b}$ *est dérivable.*

Démonstration. Nous séparons l'affirmation 1 en deux sous-cas : soit x est libre dans M, soit non. Si ce n'est pas le cas, la première condition de bonne formation sur nos jugements garantit que le type $\mathbf{a}$ de x appartient à E. Par conséquent, la seconde condition de bonne formation garantit que si $y \in \Delta$, alors $\Delta(y)$ est également dans E. Considérons alors la dérivation d'origine de M : en appliquant le lemme de renforcement pour la variable x et le lemme

d'affaiblissement pour toutes les variables de Δ, nous obtenons la dérivation de $\Gamma, \Delta \vdash M[N/x] : \mathbf{b}$ attendue. Ce sous-cas étant traité, nous supposons maintenant que $x \in \mathrm{FV}(M)$ dans le cadre de l'affirmation 1.

Nous démontrons maintenant les deux affirmations par induction sur $\mathtt{M}$, avec l'hypothèse supplémentaire suivante : si $x : \mathbf{a}$ appartient à Γ, alors $x \in \mathrm{FV}(M)$. Par commodité, nous notons $\Gamma \vdash M : \mathbf{b}$ la conclusion de $\mathtt{M}$, où x peut optionnellement appartenir au domaine de Γ. :

cst Dans ce cas $x \notin \mathrm{FV}(M)$, ce qui contredit nos hypothèses sur l'affirmation 1 ; l'affirmation 2 est vérifiée en directement par $\mathtt{M}$.

var Supposons d'abord que $M \neq x$: nous pouvons alors appliquer le même raisonnement que pour le cas précédent. Dans le cas contraire, $M = x$, et *var* garantit que $x \in \Gamma$, contredisant les hypothèses de l'affirmation 2 ; il nous suffit donc de vérifier l'affirmation 1. La seconde condition de bonne formation sur nos jugements garantit que le type de toute variable autre que x dans Γ appartient à E ; nous pouvons donc appliquer le lemme d'affaiblissement pour ces variables à la dérivation $\mathtt{N}$, produisant la dérivation attendue.

abs/**abs**, $\perp$ Dans ces cas, il suffit d'appliquer l'hypothèse d'induction à la sous-dérivation immédiate de $\mathtt{M}$, puis de ré-appliquer la dernière règle de $\mathtt{M}$ pour obtenir le résultat.

app Dans ce cas, $M = M_1 M_2$, et $\mathtt{M}$ est construite à partir des deux sous-dérivations $\mathtt{M_1} :: \Gamma_1 \vdash M_1 : \boldsymbol{c} \to \mathbf{b}$ et $\mathtt{M_2} :: \Gamma_2 \vdash M_2 : \boldsymbol{c}$. Les domaines de Γ_1, Γ_2 et Δ sont alors disjoints, et x appartient au plus à l'un des Γ_i. Nous appliquons donc l'une ou l'autre affirmation de l'hypothèse d'induction à chacune des sous-dérivations de $\mathtt{M}$, et combinons les deux dérivations résultantes à l'aide de la règle *app* pour obtenir la dérivation visée.

app, $\perp$ Dans ce cas, $M = M_1 M_2^\beta$; nous concluons en appliquant l'hypothèse d'induction à la sous-dérivation de M_1 et en combinant le résultat avec le terme $M_2[N^\alpha/x^\alpha]^\beta$ par *app*, $\perp$. □

Lemme B.10 (Réduction du sujet)**.** *Étant donné un terme affine M tel que que $\Gamma \vdash M : \mathbf{b}$ est dérivable, si $M \xrightarrow{*}_{\beta\eta} N$ alors $\Gamma \vdash N : \mathbf{b}$ est dérivable.*

Démonstration. Nous montrons ci-après que ce lemme est vérifié dans le cas où $M \to_{\beta\eta} N$. Le cas général se démontre ensuite facilement par induction sur la longueur de la réécriture, par application directe de ce résultat. Soit $\mathtt{M}$ et $\mathtt{N}$ les dérivations associées respectivement à M et N dans l'énoncé du lemme.

Considérons d'abord le cas où $M \to_\eta N$: la preuve procède par induction structurelle sur $\mathtt{M}$, et le seul cas particulier est celui où $M = \lambda x.N\ x$. La dérivation $\mathtt{N}$ est alors la sous-dérivation de $\mathtt{M}$ obtenue en considérant la branche gauche de la règle *app* qui précède la règle *abs* qui conclut la dérivation.

L'autre cas à considérer est celui où $M \to_\beta N$. Encore une fois, nous procédons par induction structurelle sur $\mathtt{M}$, où le seul cas présentant un intérêt est

celui où M est un β-redex et N son contractum. Ce cas est traité par le lemme de substitution précédent, la première affirmation correspondant au cas où $\mathtt{M}$ se conclut par *abs* et *app* et la seconde à celui où $\mathtt{M}$ se conclut par $abs, \bot$ et $app, \bot$. □

Lemme B.11 (Lemme d'extraction affine)**.** *Étant donné un terme M et une variable x tels que M contient au plus une occurrence libre de x, s'il existe une dérivation $\mathtt{D} :: \Gamma \vdash M[N/x] : \mathbf{b}$, alors l'une des deux affirmations suivantes est vraie :*

1. *Il existe une dérivation bien formée $\mathtt{M_1} :: \Gamma \vdash M : \mathbf{b}$.*
2. *OU Il existe Γ_M, Γ_N et $\mathbf{a}$, qui permettent de construire les dérivations $\mathtt{M_2} :: \Gamma_M, x : \mathbf{a} \vdash M : \mathbf{b}$ et $\mathtt{N} :: \Gamma_N \vdash N : \mathbf{a}$, et tels que $\Gamma_M, \Gamma_N = \Gamma$.*

Démonstration. La preuve procède par induction structurelle sur $\mathtt{D}$:

cst Dans ce cas la première affirmation est vraie, avec $\mathtt{M_1} = \mathtt{D}$.

var Dans ce cas $M[N/x] = y$, et soit $M = y$, soit $M = x$ et $N = y$. Le premier sous-cas se ramène au cas précédent avec $\mathtt{M_1} = \mathtt{D}$; dans le second sous cas, nous choisissions $\Gamma_M = \emptyset$, $\Gamma_N = \Gamma$ et $\mathbf{a} = \mathbf{b}$: la règle *var* produit alors immédiatement $\mathtt{M_2}$ et $\mathtt{N} = \mathtt{D}$.

abs Dans ce cas, $\mathtt{D}$ possède une sous-dérivation $\mathtt{D'} :: \Gamma, y : \boldsymbol{c} \vdash P[N/x] : \boldsymbol{d}$, avec $\mathbf{b} = \boldsymbol{c} \to \boldsymbol{d}$ et $M = \lambda y.P$. L'hypothèse d'induction appliquée à $\mathtt{D'}$ nous fournit suivant le cas :

soit une dérivation $\mathtt{M'_1} :: \Gamma, y : \boldsymbol{c} \vdash P : \boldsymbol{d}$; dans ce cas, nous appliquons la règle *abs* pour y à $\mathtt{M'_1}$, produisant directement une dérivation $\mathtt{M_1}$ qui vérifie la première affirmation ;

soit deux dérivations $\mathtt{M'_2} :: \Gamma'_M, y : \boldsymbol{c}, x : \mathbf{a} \vdash P : \boldsymbol{d}$ et $\mathtt{N'} :: \Gamma_N \vdash N : \mathbf{a}$; dans ce cas, nous vérifions la seconde affirmation en choisissant $\mathtt{N} = \mathtt{N'}$, et en construisant $\mathtt{M_2}$ à partir de $\mathtt{M'_2}$ en y appliquant la règle *abs* pour y, comme précédemment.

abs, $\bot$ Ce cas est traité de manière similaire à *abs*, avec pour seule différence qu'aucun des environnements de typage n'y assigne de type à y (et $\boldsymbol{c} = V_\alpha$).

app Dans ce cas, $M = M'M''$ et, l'hypothèse du lemme garantissant que M contient au plus une occurrence de x, la variable x est libre soit dans M', soit dans M'', soit aucun des deux. Dans ce dernier cas, la première affirmation est encore une fois immédiatement vraie avec $\mathtt{M_1} = \mathtt{D}$.

Nous vérifions maintenant le lemme dans le premier sous-cas (x est libre dans M'), le second s'ensuivant immédiatement par symétrie : $\mathtt{D}$ possède deux sous-dérivations immédiates que nous notons $\mathtt{M'} :: \Gamma' \vdash M' : \boldsymbol{c} \to \mathbf{b}$ et $\mathtt{M''} :: \Gamma'' \vdash M'' : \boldsymbol{c}$. Étant donné que x n'est pas libre dans M'', nous avons $M''[N/x] = M''$, qui est dérivable par $\mathtt{M''}$. Nous appliquons maintenant l'hypothèse d'induction à $\mathtt{M'}$ et obtenons, selon le cas, l'une des deux affirmations du lemme :

Si la première affirmation est valide, alors il existe une dérivation $\mathtt{M}'_1 :: \Gamma' \vdash M' : \boldsymbol{c} \to \mathbf{b}$. Nous combinons alors cette dérivation avec $\mathtt{M}''$ par la règle *app*, produisant une dérivation $\mathtt{M}_1$ qui satisfait la première affirmation du lemme.

Si la seconde affirmation est valide, nous obtenons deux dérivations $\mathtt{M}'_2 :: \Gamma'_M, x : \mathbf{a} \vdash M' : \boldsymbol{c} \to \mathbf{b}$ et $\mathtt{N}' :: \Gamma_N \vdash N : \mathbf{a}$. Le lemme B.7 nous permet de supposer que $x \notin \Gamma''$, par conséquent nous pouvons combiner $\mathtt{M}'_2$ et $\mathtt{M}''$ par *app* pour obtenir une dérivation $\mathtt{M}_2$. La seconde affirmation du lemme est alors satisfaite par $\mathtt{M}_2$ et $\mathtt{N} = \mathtt{N}'$.

app, $\bot$ À nouveau, nous avons $M = M'M''$; nous notons α le type syntaxique de M'' et $\mathtt{M}'$ la sous-dérivation immédiate de $\mathtt{M}$ typant M'. Nous considérons deux cas, en fonction de la présence d'une occurrence libre de x dans M' :

Soit $x \in \mathrm{FV}(M)$; nous appliquons alors l'hypothèse d'induction à $\mathtt{M}'$. Nous obtenons selon le cas soit une dérivation $\mathtt{M}'_1 :: \Gamma \vdash M' : V_\alpha \to \mathbf{b}$ qui nous permet de satisfaire la première affirmation en y appliquant la règle *app*, $\bot$; soit deux dérivations $\mathtt{M}'_2 :: \Gamma_M, x : \mathbf{a} \vdash M' : V_\alpha \to \mathbf{b}$ et $\mathtt{N}' :: \Gamma_N \vdash N : \mathbf{a}$, nous permettant de satisfaire la seconde affirmation en appliquant *app*, $\bot$ à $\mathtt{M}'_2$ pour obtenir $\mathtt{M}_2$, et en prenant $\mathtt{N} = \mathtt{N}'$.

Soit $x \notin \mathrm{FV}(M)$; ce cas est résolu en appliquant immédiatement la règle *app*, $\bot$ à $\mathtt{M}'$ pour obtenir $\mathtt{M}_1$, satisfaisant la première affirmation. Observons que le terme introduit par cette règle à droite de l'application peut être directement M'' (si $x \notin \mathrm{FV}(M'')$) ou bien une variante de M'' dans laquelle une occurrence de N a été remplacée par x : le type syntaxique α du terme introduit demeure le même dans tous les cas. □

Lemme B.12 (Expansion affine du sujet)**.** *Étant donné deux termes affines M et N tels que $M \xrightarrow{*}_{\beta\eta} N$ et un environnement de typage Γ incluant les variables libres de M, s'il existe une dérivation $\mathtt{N} :: \Gamma \vdash N : \mathbf{a}$, alors il existe une dérivation $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ telle que $\mathtt{M} \xrightarrow{*}_{\beta\eta} \mathtt{N}$.*

Démonstration. Comme pour la réduction du sujet, il nous suffit de vérifier le cas où $M \to_{\beta\eta} N$, et le résultat général s'ensuit par induction sur la longueur de la séquence de réductions.

Le cas où $M \to_\eta N$ est immédiatement résolu en appliquant les règles *var*, *app* et *abs* à l'emplacement du contractum, et ne présente pas de difficulté particulière (le terme M étant affine par hypothèse, le lemme de renforcement permet au besoin de fusionner les environnements de typage).

Le cas où $M \to_\beta N$ est vérifié par induction sur la structure de M, le seul cas particulier étant celui où $M = (\lambda x.O)\ P$ et $N = O[P/x]$, traité par le lemme d'extraction précédent. Le terme M étant par hypothèse affine, il y a au plus une occurrence de x dans O, et le résultat s'ensuit donc en appliquant le lemme B.11. □

B.3.2 Propriétés des termes ε-normaux

Nous considérons maintenant le système de réécriture $\to_\varepsilon$, dont nous rappelons les règles dans la figure B.4. Comme observé lors de sa définition, celui-ci préserve le type sémantique et l'environnement de typage de la conclusion d'une dérivation, modifiant uniquement son sujet. Nous démontrons qu'il induit une forme normale unique sur les dérivations (lemme B.13), puis établissons plusieurs propriétés des termes ε-normaux : ceux-ci sont affines (lemme B.15) et toutes leurs variables libres appartiennent à leur environnement de typage (lemme B.14). Ces deux résultats intermédiaires nous permettront d'établir par la suite d'autres résultats sur l'interaction entre $\to_\varepsilon$ et $\to_\beta$.

Lemme B.13 (Forme ε-normale)**.** *Le système de réécriture $\to_\varepsilon$ induit pour toute dérivation une unique forme normale (lemme B.13).*

Démonstration. D'une part, le système $\to_\varepsilon$ est fortement normalisant : le nombre d'occurrences de $\otimes$ ou $\odot$ dans le sujet M en conclusion de la dérivation décroît strictement par application des règles 3 à 8 (et ne croît pas par application des règles 1 et 2) ; et les deux premières règles substituent une constante (ε ou Ω) à un sous-terme distinct, limitant leur nombre d'applications.

D'autre part, les seules paires critiques de $\to_\varepsilon$ impliquent la règle 1, qui demeure applicable dans tous les cas après l'application d'une autre règle, et sont donc immédiatement joignables : par conséquent, le système $\to_\varepsilon$ est localement confluent[*cf.* Kirchner et Kirchner, 2006, p. 200].

La confluence locale et la normalisation forte du système $\to_\varepsilon$ entraînent sa confluence, ainsi que l'existence d'une forme normale unique pour tout terme M[*cf.* Kirchner et Kirchner, 2006, p. 60]. □

Lemme B.14 (Inclusion des variables libres de $|M|_\varepsilon$ dans Γ)**.** *Si la dérivation* $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ *est en forme ε-normale, alors toutes les variables libres de M sont dans le domaine de Γ.*

Démonstration. Ce résultat s'obtient par une induction simple sur $\mathtt{M}$, en considérant les règles de $\to_\varepsilon$.

Dans le cas de base, les règles *cst* et *var* satisfont la propriété. Celle-ci s'ensuit aussi immédiatement en appliquant l'hypothèse d'induction aux sous-dérivations des règles *app*, *abs* et *abs*, $\bot$. Le seul cas non-trivial est celui de la règle *app*, $\bot$, dans lequel le terme appliqué N^α peut normalement contenir n'importe quelles variables libres, indépendamment de Γ. Cependant, par hypothèse, $\mathtt{M}$ est en forme ε-normale, par conséquent il faut que $N^\alpha = \Omega^\alpha$ (le cas contraire permettrait d'appliquer la règle 2 de $\to_\varepsilon$, entraînant une contradiction). Il s'ensuit que $\mathrm{FV}(MN) = \mathrm{FV}(M)$, et cet ensemble est inclus dans $\mathrm{Dom}(\Gamma)$ par hypothèse d'induction. □

$$\frac{\mathtt{M}\,::}{\Gamma \vdash M : \mathcal{E}} \longrightarrow_\varepsilon \frac{}{\Gamma \vdash \varepsilon : \mathcal{E}}\, cst \qquad (M \neq \varepsilon) \tag{1}$$

$$\frac{\dfrac{\mathtt{M}\,::}{\Gamma \vdash M : V_\alpha \to \mathbf{b}}}{\Gamma \vdash MN^\alpha : \mathbf{b}}\, app, \bot \longrightarrow_\varepsilon \frac{\dfrac{\mathtt{M}\,::}{\Gamma \vdash M : V_\alpha \to \mathbf{b}}}{\Gamma \vdash M\Omega^\alpha : \mathbf{b}}\, app, \bot \qquad (N^\alpha \neq \Omega^\alpha) \tag{2}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{R} \to \mathcal{E} \to \mathcal{R}}\, cst \longrightarrow_\varepsilon \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{R}, y : \mathcal{E} \vdash x : \mathcal{R}}\, var}{\Gamma, x : \mathcal{R} \vdash \lambda y.x : \mathcal{E} \to \mathcal{R}}\, abs}{\Gamma \vdash \lambda xy.x : \mathcal{R} \to \mathcal{E} \to \mathcal{R}}\, abs \tag{3}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{E} \to \mathcal{R} \to \mathcal{R}}\, cst \longrightarrow_\varepsilon \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{R} \vdash y : \mathcal{R}}\, var}{\Gamma, x : \mathcal{E} \vdash \lambda y.y : \mathcal{R} \to \mathcal{R}}\, abs}{\Gamma \vdash \lambda xy.y : \mathcal{E} \to \mathcal{R} \to \mathcal{R}}\, abs \tag{4}$$

$$\frac{}{\Gamma \vdash \odot : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, cst \longrightarrow_\varepsilon \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{E} \vdash \varepsilon : \mathcal{E}}\, cst}{\Gamma, x : \mathcal{E} \vdash \lambda y.\varepsilon : \mathcal{E} \to \mathcal{E}}\, abs}{\Gamma \vdash \lambda xy.\varepsilon : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, abs \tag{5}$$

$$\frac{}{\Gamma \vdash \otimes : \mathbf{a} \to \mathcal{E} \to \mathbf{a}}\, cst \longrightarrow_\varepsilon \frac{\dfrac{\dfrac{}{\Gamma, x : \mathbf{a}, y : \mathcal{E} \vdash x : \mathbf{a}}\, var}{\Gamma, x : \mathbf{a} \vdash \lambda y.x : \mathcal{E} \to \mathbf{a}}\, abs}{\Gamma \vdash \lambda xy.x : \mathbf{a} \to \mathcal{E} \to \mathbf{a}}\, abs \qquad (\mathbf{a} \neq \mathcal{E}) \tag{6}$$

$$\frac{}{\Gamma \vdash \otimes : \mathcal{E} \to \mathbf{a} \to \mathbf{a}}\, cst \longrightarrow_\varepsilon \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathbf{a} \vdash y : \mathbf{a}}\, var}{\Gamma, x : \mathcal{E} \vdash \lambda y.y : \mathbf{a} \to \mathbf{a}}\, abs}{\Gamma \vdash \lambda xy.y : \mathcal{E} \to \mathbf{a} \to \mathbf{a}}\, abs \qquad (\mathbf{a} \neq \mathcal{E}) \tag{7}$$

$$\frac{}{\Gamma \vdash \otimes : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, cst \longrightarrow_\varepsilon \frac{\dfrac{\dfrac{}{\Gamma, x : \mathcal{E}, y : \mathcal{E} \vdash \varepsilon : \mathcal{E}}\, cst}{\Gamma, x : \mathcal{E} \vdash \lambda y.\varepsilon : \mathcal{E} \to \mathcal{E}}\, abs}{\Gamma \vdash \lambda xy.\varepsilon : \mathcal{E} \to \mathcal{E} \to \mathcal{E}}\, abs \tag{8}$$

FIGURE B.4 – Rappel des règles de $\rightarrow_\varepsilon$

Lemme B.15 (Affinité de $|M|_\varepsilon$)**.** *Si* $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ *est une dérivation en forme* ε*-normale, alors le terme* M *est affine.*

Démonstration. Par définition, un terme M est affine si et seulement si aucun sous-terme de M ne contient deux occurrence libres d'une même variable. Nous montrons cette propriété par induction sur la structure de M, en profitant du fait que la structure d'une dérivation reflète directement celle de son sujet. Le seul cas présentant un intérêt est celui où $M = M_1 M_2$ est une application ; deux sous-cas sont possibles, en fonction de la règle employée pour typer M dans $\mathtt{M}$ (*app* ou *app*, $\bot$).

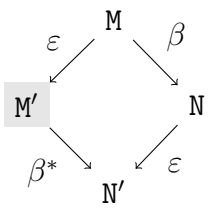
D'une part, si la règle correspondante dans $\mathtt{M}$ est *app*, $\bot$, alors le jugement $\Gamma \vdash M_1 : V_\alpha \to \mathbf{a}$ est dérivable et, puisque $\mathtt{M}$ est en forme ε-normale, il faut que $M_2 = \Omega^\alpha$ (le contraire entraînerait une contradiction, par application de la règle 2 de $\to_\varepsilon$). Par hypothèse d'induction, M_1 est affine et, par conséquent, $M = M_1\Omega^\alpha$ est également affine.

D'autre part, si la règle associée à M dans $\mathtt{M}$ est *app*, alors nous savons que $\Gamma = \Gamma_1, \Gamma_2$ tel que $\Gamma_1 \vdash M_1 : \mathbf{b} \to \mathbf{a}$ et $\Gamma_2 \vdash M_2 : \mathbf{b}$ pour un type $\mathbf{b}$ donné ; les domaines de Γ_1 et Γ_2 étant disjoints. Par hypothèse d'induction, M_1 et M_2 sont affines, et nous savons que les sous-dérivation correspondantes sont en forme ε-normale : le lemme B.14 entraîne alors que les variables libres de M_1 (resp. M_2) sont incluses dans le domaine de Γ_1 (resp. Γ_2). Ces deux domaines étant disjoints, nous pouvons conclure que $\mathrm{FV}(M_1) \cap \mathrm{FV}(M_2) = \emptyset$, montrant finalement que le terme $M = M_1 M_2$ est affine. □

B.3.3 Interaction de $\to_\beta / \to_\varepsilon$

Nous donnons dans cette section la preuve de la propriété de réordonnement de $\to_\beta / \to_\varepsilon$. Nous établissons ensuite plusieurs résultats intermédiaires pour montrer finalement l'existence d'une forme $\beta\varepsilon$-normale unique sur les dérivations.

Lemme B.16 (Réordonnement de $\to_\beta / \to_\varepsilon$)**.** *Considérons deux termes affines* M *et* N *et leurs dérivations associées* $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ *et* $\mathtt{N} :: \Gamma \vdash N : \mathbf{a}$*, telles que* $\mathtt{M} \to_\beta \mathtt{N}$ *selon la procédure du lemme B.10. Si* $\mathtt{N} \to_\varepsilon \mathtt{N}' :: \Gamma \vdash N' : \mathbf{a}$*, alors il existe un terme* M' *muni d'une dérivation* $\mathtt{M}' :: \Gamma \vdash M' : \mathbf{a}$ *de sorte que* $\mathtt{M} \to_\varepsilon \mathtt{M}'$ *et* $\mathtt{M}' \overset{*}{\to}_\beta \mathtt{N}'$*. Cette relation est illustrée par le diagramme suivant :*



Démonstration. En exploitant le fait que la structure d'une dérivation reflète directement celle de son sujet, nous considérons la position relative des termes réécrits par $\to_\beta$ et $\to_\varepsilon$ dans N, et distinguons trois cas :

— Si la position où a lieu la réécriture $\to_\varepsilon$ domine strictement celle de $\to_\beta$, la règle utilisée pour $\to_\varepsilon$ doit être la première ou la seconde (les règles 3 à 8 portent sur des constantes, et ne peuvent donc pas dominer de sous-terme) ; nous pouvons alors appliquer alors cette même règle dans terme M pour obtenir M', et distinguer deux cas :

soit le β-redex transformant M en N est effacé dans M' (le terme qui le contient ayant été remplacé par ε par la règle 1, ou par Ω dans la règle 2), auquel cas $M' = N'$, et nous pouvons donc choisir $\mathtt{N}'$ comme dérivation pour M' ;

soit le β-redex transformant M en N appartient au sous-terme gauche de la règle 2, qui est inchangé par ε-réduction, auquel cas la même β-contraction peut être effectuée dans M' pour obtenir N'.

— Si les positions respectives des réécritures $\to_\varepsilon$ et $\to_\beta$ sont incomparables, alors les redex sont indépendants ; l'ordre de $\to_\beta$ et $\to_\varepsilon$ peut dans ce cas être inversé sans problème.

— Enfin, si la position de la réécriture $\to_\beta$ domine celle de $\to_\varepsilon$, le terme M' et sa dérivation peuvent être obtenus de la même façon que précédemment. En effet, considérons le β-redex R contracté pour passer de M à N, que nous notons $R = (\lambda x.P)Q$; deux types de configuration se présentent alors :

soit le redex ε dans le sous-terme $P[Q/x]$ de N ne domine pas le terme Q substitué, et nous pouvons inverser l'ordre de $\to_\beta$ et $\to_\varepsilon$ sans interaction particulière entre les deux redex ;

soit enfin le redex domine le terme substitué dans P, et la contraction correspondante par $\to_\varepsilon$ dans M est susceptible d'effacer l'occurrence (unique) de la variable x ; le terme résultant peut néanmoins toujours être typé de sorte que sa β-contraction produise la dérivation $\mathtt{N}'$ (en particulier, la variable x doit dans ce cas avoir un type de E).

Un terme M' correspondant au diagramme ci-dessus peut donc toujours être construit. Observons cependant que la condition que M et N sont des termes affines est requise dans l'énoncé actuel du lemme (le cas échéant, plusieurs réductions $\to_\varepsilon$ supplémentaires pourraient être requises pour convertir M' en N'). □

Lemme B.17 (Normalisation forte de $\to_{\beta\varepsilon}$)**.** *Il n'existe pas de séquence infinie de dérivations* $(\mathtt{M_i} :: \Gamma \vdash M_i : \mathbf{a})_{i\in\mathbb{N}}$ *telle que* $\mathtt{M_i} \to_{\beta\varepsilon} \mathtt{M_{i+1}}$.

Démonstration. Nous prouvons ce lemme par contradiction. Supposons qu'il existe une telle séquence : la preuve du lemme B.13 établit la normalisation forte du système $\to_\varepsilon$, interdisant la présence d'une séquence infinie de réductions par $\to_\varepsilon$ à partir de $\mathtt{M_0}$. Or, l'application itérée du lemme B.16 nous as-

sure que toute réduction par $\to_\varepsilon$ peut être réalisée avant toute réduction par $\to_\beta$: par conséquent, la séquence de réductions qui produit $(\mathtt{M_i})$ doit s'achever par une sous-séquence infinie de β-réductions. Cependant, cette dernière conclusion contredit la propriété de normalisation forte du λ-calcul simplement typé, montrant l'impossibilité de l'existence d'une séquence infinie de β/ε-réductions. □

Lemme B.18 (Réduction par $\to_\varepsilon$ modulo substitution)**.** *Soit* $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ *une dérivation. Si* $\mathtt{M} \xrightarrow{*}_\varepsilon \mathtt{M'} :: \Gamma \vdash M' : \mathbf{a}$ *de sorte que* $x \notin \mathrm{FV}(M')$*, et si* $\mathtt{D} :: \Gamma \vdash M[N/x] : \mathbf{a}$ *est la dérivation obtenue à partir de* M *en appliquant le lemme de substitution (lemme B.9), alors* $\mathtt{D} \xrightarrow{*}_\varepsilon \mathtt{M'}$*.*

Démonstration. Nous procédons par induction sur la structure de M :

cst Dans ce cas, $\mathtt{M} = \mathtt{D}$, et le résultat est immédiat.

var Nous considérons deux sous-cas. Si $M \neq x$, alors $\mathtt{M} = \mathtt{D}$ et le résultat est à nouveau immédiat. Dans le cas contraire, puisque x n'est, par hypothèse, pas libre dans M', il s'ensuit nécessairement que $\mathbf{a} = \mathcal{E}$ et $M' = \varepsilon$. Par conséquent, nous pouvons déduire que $\mathtt{D} :: \Gamma \vdash N : \mathcal{E}$ et que donc $\mathtt{D} \xrightarrow{*}_\varepsilon \mathtt{M'} :: \Gamma \vdash \varepsilon : \mathcal{E}$.

abs/abs, $\bot$ Dans ces cas, tous les ε-redex de M sont contenus dans sa sous-dérivation immédiate (aucun ε-redex ne se conclut par *abs* ou $abs, \bot$). Donc, en appliquant à celle-ci l'hypothèse d'induction, nous obtenons une séquence de réductions réécrivant la sous-dérivation immédiate de D en celle de M′ ; cette séquence demeure valide en ajoutant une occurrence de *abs* ou de $abs, \bot$ pour obtenir une réduction de D en M′.

app Nous considérons à nouveau deux sous-cas. D'une part, la dernière instance de *app* dans M peut appartenir à un ε-redex contracté dans M′ : la seule règle du système $\to_\varepsilon$ qui le permette est la première, ce qui implique à nouveau que $\mathbf{a} = \mathcal{E}$ et $M' = \varepsilon$, permettant à D de se réécrire en M′ en une étape. Dans le cas contraire, la dernière instance de *app* dans M se retrouve à la fois dans M′ et D, et chacune de ces dérivations possède deux sous-dérivations immédiates, auxquelles nous pouvons appliquer l'hypothèse d'induction. Il en résulte deux chaînes de réductions par $\to_\varepsilon$ permettant de réécrire les sous-dérivations de D en sous-dérivation de M′. Nous pouvons alors combiner ces séquences de réductions dans D pour obtenir M′, comme précédemment.

app, $\bot$ Nous distinguons à nouveau le cas où la dernière instance de $app, \bot$ dans M appartient à un redex contracté dans M′ : si tel est le cas, la règle de $\to_\varepsilon$ employée est soit la première (auquel cas le résultat s'ensuit comme dans le cas précédent), soit la seconde. Dans ce dernier cas, nous pouvons appliquer l'hypothèse d'induction à la sous-dérivation immédiate de M et obtenir le résultat en contractant D de la même manière que précédemment. Il se peut également que l'occurrence de $app, \bot$ qui conclut M ne fasse pas partie d'un ε-redex contracté dans M′. Dans ce

dernier cas, nous obtenons une fois encore le résultat en appliquant l'hypothèse d'induction à la sous-dérivation immédiate de $\mathtt{M}$ (en observant que par hypothèse que x n'est pas libre dans le terme introduit à droite par $app, \bot$). □

Lemme B.19 (Forme $\beta\varepsilon$-normale)**.** *Le système de réécriture $\to_{\beta\varepsilon}$ induit une forme normale unique.*

Démonstration. Le lemme B.17 établit la normalisation forte du système $\to_{\beta\varepsilon}$, il suffit par conséquent de montrer sa confluence locale pour établir qu'il est confluent (et induit donc une forme normale unique). Nous montrons donc ici la confluence locale, en listant les paires critiques du système et en prouvant leur joignabilité. Nous avons montré dans la preuve du lemme B.13 que toutes les paires critiques de $\to_\varepsilon$ sont joignables, et la confluence de $\to_\beta$ est connue. Deux paires critiques restent alors à considérer, entre $\to_\beta$ et les deux premières règles de $\to_\varepsilon$.

Nous considérons d'abord la paire critique formée par $\to_\beta$ et la première règle de $\to_\varepsilon$: ces deux réécritures modifient la dérivation d'un jugement $\Gamma \vdash M : \mathcal{E}$ où M est un β-redex. Ces deux règles ne modifient cependant que le sujet du jugement final, préservant son type et son environnement ; par conséquent, la première règle de $\to_\varepsilon$ est toujours applicable après $\to_\beta$, sauf dans le cas où $M \to_\beta \varepsilon$ (auquel cas la paire critique est jointe immédiatement).

L'autre paire critique à considérer est séparée par $\to_\beta$ et la deuxième règle de $\to_\varepsilon$. Dans ce cas, nous disposons d'une dérivation $\mathtt{D} :: \Gamma \vdash (\lambda x.M)N^\alpha : \mathbf{a}$; en raison de la forme des règles de notre système de typage, $\mathtt{D}$ possède deux sous-dérivations successives, notées $\mathtt{D}' :: \Gamma \vdash \lambda x.M : V_\alpha \to \mathbf{a}$ (sous la règle $app, \bot$) et $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ (sous la règle $abs, \bot$). D'après le lemme B.13, $\mathtt{M}$ possède une forme ε-normale, que nous notons $\mathtt{M}_{\mathtt{FN}} :: \Gamma \vdash M' : \mathbf{a}$; par application du lemme B.14, le sujet M' de cette dernière dérivation est tel que $x \notin \mathrm{FV}(M')$.

Considérons maintenant la paire critique elle-même, obtenue à partir de $\mathtt{D}$ d'un côté par la dérivation $\mathtt{D} \to_\beta \mathtt{D}_\beta :: \Gamma \vdash M[N^\alpha/x] : \mathbf{a}$, et de l'autre par $\mathtt{D} \to_\varepsilon \mathtt{D}_\varepsilon :: \Gamma \vdash (\lambda x.M)\Omega^\alpha : \mathbf{a}$. Cette dernière dérivation se β-réduit à son tour en $\mathtt{D}_\varepsilon \to_\beta \mathtt{D}_{\varepsilon\beta} :: \Gamma \vdash M[\Omega^\alpha/x] : \mathbf{a}$. Nous appliquons maintenant le lemme B.18 à $\mathtt{M} \overset{*}{\to}_\varepsilon \mathtt{M}_{\mathtt{FN}}$ et $\mathtt{D}_\beta$, en observant que $x \notin \mathrm{FV}(M')$ et que $\mathtt{D}_\beta$ est obtenue par le lemme de substitution : nous en concluons que $\mathtt{D}_\beta \overset{*}{\to}_\varepsilon \mathtt{M}_{\mathtt{FN}}$. Suivant le même raisonnement, nous pouvons également établir que $\mathtt{D}_{\varepsilon\beta} \overset{*}{\to}_\varepsilon \mathtt{M}_{\mathtt{FN}}$. Par conséquent, puisque $\mathtt{D}_\varepsilon \to_\beta \mathtt{D}_{\varepsilon\beta}$, nous disposons d'une stratégie permettant de joindre la paire critique $(\mathtt{D}_\beta, \mathtt{D}_\varepsilon)$ séparée par $\to_\beta$ et la deuxième règle de $\to_\varepsilon$, nous permettant de conclure le lemme. □

B.3.4 Contraintes sur la taille des termes

Nous établissons ici plusieurs résultats successifs utilisés dans la preuve du dernier lemme, et montrons que la taille d'un terme compressé par $\to_\varepsilon$ est

bornée par la longueur des mots de son langage (ce qui nous permet notamment d'établir la complexité de la seconde phase de l'algorithme d'analyse général). Nous rappelons préalablement le contenu de la définition 5.8, portant sur la taille (concrète) d'un lambda-terme et sur la taille d'un type.

La *taille* $|M|$ d'un lambda-terme M est définie inductivement comme suit :
- $|x| = 1$
- $|c| = 1$ pour toute constante $c \in C$
- $|\lambda x.P| = |P|$
- $|MN| = |M| + |N|$

La *taille concrète* $\|M\|$ d'un lambda-terme M représentant un terme commutatif est définie inductivement comme suit :
- $\|x\| = 0$
- $\|c\| = \begin{cases} 3 \text{ si } c \in \{\odot, \otimes\} \\ 1 \text{ si } c \in \Sigma \\ 0 \text{ si } c \in \{\varepsilon, \Omega^\alpha\} \end{cases}$
- $\|\lambda x.P\| = \|P\|$
- $\|MN\| = \|M\| + \|N\|$

La *taille* $|\tau|$ d'un type syntaxique τ est définie inductivement comme suit :
- $|0| = 1$
- $|\alpha \to \beta| = |\alpha| + |\beta|$

Lemme B.20 (Non-trivialité d'un terme $|M|_\varepsilon$ avec constantes)**.** *Si une dérivation* $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ *est en forme ε-normale et que M contient au moins une constante de $\Sigma \cup \{\odot, \otimes\}$, alors* $\mathbf{a} \notin E$.

Démonstration. Nous procédons par induction sur $\mathtt{M}$ en exploitant la structure des règles de typage :

var Ce cas est exclu, puisque M contient une constante par hypothèse.

cst Dans ce cas, nous savons par hypothèse que $c \in \Sigma \cup \{\odot, \otimes\}$. Si $c \in \Sigma$, alors $\mathbf{a} = \mathcal{R}$. Dans le cas contraire, puisque M est en forme ε-normale, alors $\mathbf{a} = \boldsymbol{b} \to \boldsymbol{c} \to \mathcal{R}$. Par conséquent, dans tous les cas, $\mathbf{a} \notin E$.

abs/**abs**, $\perp$ Dans ce cas, la conclusion est une conséquence immédiate de l'hypothèse d'induction.

app Dans ce cas, $M = M_1 M_2$, et il existe un type $\mathbf{b}$ tel que les sous-dérivations de $\mathtt{M}$ sont $\mathtt{M_1} :: \Gamma_1 \vdash M_1 : \mathbf{b} \to \mathbf{a}$ et $\mathtt{M_2} :: \Gamma_2 \vdash M_2 : \mathbf{b}$. Puisque M contient au moins une constante c de $\Sigma \cup \{\odot, \otimes\}$, cette constante doit être présente dans M_1 ou M_2. Suivant le cas, l'hypothèse d'induction montre que soit $(\mathbf{b} \to \mathbf{a}) \notin E$ (si $c \in M_1$), soit $\mathbf{b} \notin E$ (si $c \in M_2$). Dans les deux cas, il s'ensuit que $\mathbf{a} \notin E$.

app, $\perp$ Dans ce dernier cas, la conclusion s'ensuit comme dans le cas précédent avec $c \in M_1$; en effet, le fait que $\mathtt{M}$ est ε-normale entraîne que $M_2 = \Omega^\alpha$. □

Lemme B.21 (Invariance de la taille concrète de $|M|_\varepsilon$ par $\to_\beta$)**.** *Si une dérivation* $\mathtt{M} :: \Gamma \vdash M : \mathbf{a}$ *est ε-normale et que $M \overset{*}{\to}_\beta N$, alors $\|M\| = \|N\|$.*

Démonstration. Supposons que $M \to_\beta N$: le résultat du lemme s'ensuit immédiatement par induction sur la longueur de la β-réduction ; nous montrons donc le cas restreint où $M \to_\beta N$, par induction sur $\mathtt{M}$. Le seul cas présentant un intérêt est celui où M est le β-redex $(\lambda x.M')P$ et $N = M'[P/x]$. La dernière règle utilisée dans $\mathtt{M}$ est alors *app* ou *app*, $\bot$. Dans ce dernier cas, puisque $\mathtt{M}$ est en forme ε-normale, $P = \Omega^\alpha$; or $\|\Omega^\alpha\| = \|x\| = 0$, par conséquent nous obtenons immédiatement que $\|M\| = \|M'\| = \|N\|$.

Dans le cas où la dernière règle utilisée est *app*, puisque M est affine d'après le lemme B.15, M' contient au plus une occurrence de x. Si x apparaît une fois dans M', le résultat s'ensuit immédiatement, puisque $\|x\| = 0$. Dans le cas contraire, $x \notin \mathrm{FV}(M')$: il existe alors un type $\mathbf{b}$ tel que les sous-dérivation de $\mathtt{M}$ sont $\mathtt{M_1} :: \Gamma_1 \vdash \lambda x.M' : \mathbf{b} \to \mathbf{a}$ et $\mathtt{M_2} :: \Gamma_2 \vdash P : \mathbf{b}$. La dérivation $\mathtt{M_1}$ est elle-même construite au moyen de la règle *abs* à partir du jugement suivant : $\Gamma_1, x : \mathbf{b} \vdash M' : \mathbf{a}$; or nous savons que x n'est pas libre dans M', par conséquent, la première clause de la définition des jugements bien formés impose que $\mathbf{b} \in E$. La contraposée du lemme B.20 impose alors que $\|P\| = 0$ (puisque le type de P appartient à E, P ne peut contenir aucune constante de $\Sigma \cup \{\odot, \otimes\}$), et il s'ensuit à nouveau que $\|M\| = \|M'\| = \|N\|$. □

Lemme B.22 (Contrainte sur la taille des termes)**.** *Étant donné un terme affine typable M^α en forme β-normale, si $\|M\| = 0$ et que $x_1^{\alpha_1} \dots x_n^{\alpha_n}$ sont les variables libres de M, alors l'inégalité suivante est vérifiée :*

$$|M| \leq |\alpha| + \sum_{i=1}^{n} |\alpha_i|$$

Démonstration. Nous montrons ce lemme par induction sur M :

— Si M^α est une constante, le résultat est immédiat puisque $|\alpha| > 0$ et que, par hypothèse, $M \in \{\varepsilon, \Omega^\alpha\}$.
— Si M^α est une abstraction, nous avons $\alpha = \beta \to \gamma$ et $M = \lambda x^\beta.P^\gamma$; en outre, $|M| = |P|$. Par hypothèse d'induction, si $x \notin \mathrm{FV}(P)$, le résultat

est immédiat. Dans le cas contraire, nous obtenons :

$$\begin{aligned}|M| = |P| &\leq |\gamma| + \sum_{x^\tau}^{\mathrm{FV}(P)} |\tau| \\ &\leq |\gamma| + |\beta| - |\beta| + \sum_{x^\tau}^{\mathrm{FV}(P)} |\tau| \\ &\leq |\beta \to \gamma| - |\beta| + \sum_{x^\tau}^{\mathrm{FV}(P)} |\tau| \\ &\leq |\beta \to \gamma| + \sum_{x^\tau}^{\mathrm{FV}(M)} |\tau|\end{aligned}$$

la dernière ligne étant justifiée par le fait que x^β est libre dans P mais pas dans M.

— Enfin, si $M^\alpha = x^\delta M_1^{\gamma_1} \dots M_m^{\gamma_m}$ (ce qui couvre tous les cas restants car M est en forme β-normale par hypothèse), alors nous supposons sans perte de généralité que x^δ est la première variable libre $x_1^{\alpha_1}$ de M, et que donc $\alpha_1 = \gamma_1 \to \dots \to \gamma_m \to \alpha$. Pour tout $j \in [m]$, $\mathrm{FV}(M_j)$ dénote l'ensemble des variables libres de M_j et, puisque M est affine par hypothèse, les ensembles $\{x_1^{\alpha_1}\}, \mathrm{FV}(M_1), \dots, \mathrm{FV}(M_m)$ forment une partition de $\mathrm{FV}(M)$. Par conséquent nous avons l'égalité :

$$\sum_{i=1}^{n} |\alpha_i| = |\alpha_1| + \sum_{j=1}^{m} \sum_{y^\beta}^{\mathrm{FV}(M_j)} |\beta| \tag{B.1}$$

En outre, par hypothèse d'induction, sous savons que pour tout $j \in [m]$:

$$|M_j| \leq |\gamma_j| + \sum_{y^\beta}^{\mathrm{FV}(M_j)} |\beta| \tag{B.2}$$

Comme $|M| = 1 + |M_1| + \dots + |M_m|$, nous obtenons par (B.2) :

$$|M| \leq 1 + \sum_{j=1}^{m} \left(|\gamma_j| + \sum_{y^\beta}^{\mathrm{FV}(M_j)} |\beta| \right)$$

et par suite, en appliquant (B.1) :

$$|M| \leq 1 + \sum_{j=1}^{m} |\gamma_j| - |\alpha_1| + \sum_{i=1}^{n} |\alpha_i|$$

Enfin, puisque $|\alpha_1| = |\alpha| + \sum_{j=1}^{m} |\gamma_j|$, nous obtenons :

$$|M| \leq 1 - |\alpha| + \sum_{i=1}^{n} |\alpha_i|$$

et comme $1 - |\alpha| < |\alpha|$ dans tous les cas, nous pouvons en conclure que l'inégalité du lemme est respectée. □

Corollaire B.23. *Si M^α est un terme affine typable en forme β-normale avec $\mathrm{FV}(M) = \{x_1^{\alpha_1} \dots x_n^{\alpha_n}\}$, alors sa taille $|M|$ est bornée linéairement par l'expression $\|M\| + |\alpha| + \sum_{i=1}^{n} |\alpha_i|$*

Démonstration. Par définition, $\|c\| \leq |\tau(c)|$ pour toute constante c. Par conséquent, il suffit de remplacer les constantes de M par des variables libres du même type pour établir la borne souhaitée en appliquant directement le lemme précédent. □

Corollaire B.24. *Si* `M` *est une dérivation en forme $\beta\varepsilon$-normale typant un terme clos M de type α, alors $|M| \leq \|M\| + |\alpha|$.*

Démonstration. Ce corollaire est une conséquence immédiate du précédent. □

Lemme B.25. *Soit $M, N, N_1, \dots, N_n$ des termes respectant ces conditions :*
- $\mathrm{FV}(M) = \{x_1 \dots x_n\}$
- $M[N_i/x_i] \overset{*}{\to}_\beta N$
- *Il existe des dérivation ε-normales bien formées des jugements suivants :*

$$\Gamma, x_1 : \mathbf{a_1} \dots x_n : \mathbf{a_n} \vdash M : \mathbf{b} \qquad \vdash N : \mathbf{b} \qquad \vdash N_1 : \mathbf{a_1} \ \dots \ \vdash N_n : \mathbf{a_n}$$

L'égalité suivante est respectée :

$$\sum_{i=1}^{n} \|N_i\| = \|N\| - \|M\|$$

Démonstration. Nous montrons cette égalité en construisant une dérivation ε-normale pour le terme $M[N_i/x_i]$ (dont la taille concrète est connue) et en exploitant la propriété que la taille concrète d'un terme ε-normal est préservée par β-réduction (lemme B.21).

Considérons les dérivations ε-normales `M` et $\mathtt{N_i}$ associées aux termes M et N_i, données par l'énoncé du lemme. En appliquant n fois la règle *abs* à `M`, puis en combinant la dérivation résultante avec chaque $\mathtt{N_i}$ en utilisant n fois la règle *app*, nous obtenons une dérivation `D` pour le terme $(\lambda x_1 \dots x_n.M)\ N_1 \dots N_n$; cette dérivation est en forme ε-normale (en raison de l'ε-normalité de ses sous-dérivations et des règles utilisées pour les combiner).

Par définition, cette dérivation se β-réduit en une dérivation $\mathtt{D}'$ pour le terme $M[N_i/x_i]$, qui est également en forme ε-normale : en effet, d'après le lemme B.16, si une dérivation $\mathtt{D}'$ obtenue par β-réduction de $\mathtt{D}$ contient un ε-redex, alors il est possible de réduire $\mathtt{D}$ par $\to_\varepsilon$; ce qui contredirait ici l'ε-normalité de $\mathtt{D}$.

Par suite, la dérivation $\mathtt{D}'$ se β-réduit en une dérivation $\mathtt{N}$ de N (par hypothèse du lemme). La β-réduction d'un terme ε-normal préservant sa taille concrète (d'après le lemme B.21), nous avons donc l'égalité $\|M[N_i/x_i]\| = \|N\|$. Par ailleurs, il est immédiat que $\|M[N_i/x_i]\| = \|M\| + \sum_{i=1}^{n} \|N_i\|$, ce qui équivaut à $\sum_{i=1}^{n} \|N_i\| = \|M[N_i/x_i]\| - \|M\|$, nous permettant de conclure le lemme en appliquant l'égalité précédente. □

B.3.5 Preuve de l'algorithme d'analyse général

Nous prouvons maintenant la correction, la complétude et la complexité des deux phases de l'algorithme d'analyse général donné à l'issue du chapitre 5.

Pour mémoire, celui-ci procède en établissant la solvabilité d'un ensemble de buts, qui associent un symbole non-terminal à un tuple de termes. Un but est par définition solvable si le tuple de termes qu'il contient appartient (modulo ε-compression) au langage de son non-terminal. La première phase de l'algorithme calcule l'ensemble des buts triviaux (ceux dont la taille concrète est nulle) solvables d'après la grammaire – indépendamment de w. La seconde construit un témoin (ε-normal) de l'existence d'un terme dans $\mathcal{T}(G)$ reconnaissant w ; et procède en décomposant des buts en sous-buts de taille égale ou inférieure (par le lemme B.25) en suivant les productions de G, et en éliminant automatiquement les buts triviaux identifiés comme solvables lors de la première phase.

Nous démontrons d'abord la correction et la complétude de l'algorithme dans sa première phase (décrite page 149). La correction et la complétude de la seconde phase (détaillée page 150) peuvent être établies sensiblement de la même manière, en remplaçant l'appartenance d'un but à l'ensemble inductif $\mathscr{T}(G)$ par la propriété d'être effaçable.

Théorème B.26 (Correction et complétude de la première phase)**.** *Tout but trivial $A(\mathtt{D_1}, \ldots, \mathtt{D_n})$ est solvable si et seulement si il appartient à $\mathscr{T}(G)$.*

Démonstration. Nous montrons d'abord l'implication directe. Par définition, si un but $A(\mathtt{D_1}, \ldots, \mathtt{D_n})$ est solvable, alors il existe un tuple $(P_1, \ldots, P_n)$ appartenant à $\mathcal{L}(A)$, tel que pour tout i, si $\mathtt{D_i} \neq \bot^{\alpha_i}$, alors $P_i \in \mathcal{L}(\mathtt{D_i})$. En raison de l'appartenance des P_i à $\mathcal{L}(A)$, il existe une règle r de G de la forme :

$$r = A(M_1^{\alpha_1}, \ldots, M_n^{\alpha_n}) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}) \ldots B_p(x_{p,1}, \ldots, x_{p,n_p})$$

et p tuples $\left(N_{j,1}, \ldots, N_{j,n_j}\right) \in \mathcal{L}(B_j)$, tels que $M_i[x_{j,k} \leftarrow N_{j,k}] \overset{*}{\to}_{\beta\eta} P_i$ pour tout i. Nous procédons alors par induction sur la dérivation du tuple $(P_1, \ldots, P_n)$

selon G, nous permettant de supposer que tout but trivial reposant sur un tuple de termes construit par une sous-dérivation stricte de celle des P_i appartient à l'ensemble $\mathscr{T}(G)$.

Si $\mathtt{D_i}$ est défini, il dérive $\mathtt{D_i} :: \vdash Q_i : \boldsymbol{a_i}$ et son langage contient P_i. Il existe donc une dérivation $\mathtt{P_i} :: \vdash P_i : \boldsymbol{a_i}$ et, puisque P_i est en forme $\beta\eta$-normale, $\mathtt{P_i} \xrightarrow{*}_\varepsilon \mathtt{D_i}$. De plus, puisque $M_i[x_{j,k} \leftarrow N_{j,k}] \xrightarrow{*}_{\beta\eta} P_i$, nous pouvons construire d'après le lemme B.12 une dérivation $\mathtt{S_i} :: \vdash M_i[x_{j,k} \leftarrow N_{j,k}] : \boldsymbol{a_i}$ du terme M_i après substitution, à partir de laquelle nous pouvons obtenir, par des applications itérées du lemme d'extraction (B.11), les dérivations suivantes :

$$\begin{gathered}\mathtt{M_i} :: y_{i,1} : \boldsymbol{b_{i,1}} \dots y_{i,m_i} : \boldsymbol{b_{i,m_i}} \vdash M_i : \boldsymbol{a_i} \\ \mathtt{R_{i,1}} :: \vdash R_{i,1} : \boldsymbol{b_{i,1}} \quad ; \quad \dots \quad ; \quad \mathtt{R_{i,m_i}} :: \vdash R_{i,m_i} : \boldsymbol{b_{i,m_i}}\end{gathered}$$

où l'ensemble $Y_i = \{y_{i,1} \dots y_{i,m_i}\}$ est inclus dans l'ensemble $\mathcal{X} = \{x_{j,k}\}$ des variables apparaissant dans r. Nous utilisons dans la suite par convention la variable l pour itérer sur l'ensemble $[m_i]$. Les variables $y_{i,l}$ correspondent en pratique aux variables $x_{j,k}$ qui apparaissent effectivement dans l'un des M_i qui forment le but actuel ; aussi, lorsque $y_{i,l} = x_{j,k}$, le terme $R_{i,l}$ est en fait le terme $N_{j,k}$. Nous considérons maintenant les formes ε-normales des dérivations extraites ci-dessus, que nous désignons comme suit :

$$\begin{gathered}\mathtt{M'_i} :: y_{i,1} : \boldsymbol{b_{i,1}} \dots y_{i,m_i} : \boldsymbol{b_{i,m_i}} \vdash M'_i : \boldsymbol{a_i} \\ \mathtt{R'_{i,1}} :: \vdash R'_{i,1} : \boldsymbol{b_{i,1}} \quad ; \quad \dots \quad ; \quad \mathtt{R'_{i,m_i}} :: \vdash R'_{i,m_i} : \boldsymbol{b_{i,m_i}}\end{gathered}$$

Considérons la dérivation $\mathtt{S'_i} :: \vdash M'_i\big[y_{i,l} \leftarrow R'_{i,l}\big]$ obtenue en appliquant le lemme de substitution aux variables de Y_i dans $\mathtt{M'_i}$; d'après le lemme B.18 $\mathtt{S'_i}$ est la forme ε-normale de $\mathtt{S_i}$. Ceci implique, d'après le lemme B.16 que $\mathtt{S'_i}$ a pour forme β-normale la dérivation $\mathtt{D_i}$, également ε-normale ; le lemme B.25 implique alors que $\|Q_i\| = \|M'_i\| + \sum_{l=1}^{m_i} \big\|R'_{i,l}\big\|$. Or, puisque $\mathtt{D_i}$ est triviale, nous avons $\|Q_i\| = 0$, et la taille concrète des $R'_{i,l}$ est donc également nulle. Par conséquent, les dérivations $\mathtt{R'_{i,l}}$ sont également triviales.

Finalement, nous construisions p buts $B_j(\mathtt{D'_{j,1}}, \dots, \mathtt{D'_{j,n_j}})$ où chaque élément $\mathtt{D'_{j,k}}$ est défini comme $\mathtt{R'_{i,l}}$ lorsque $x_{j,k} = y_{i,l}$, et est indéfini lorsqu'aucun des Y_i ne contient $x_{j,k}$. Nous montrons maintenant que ces buts appartiennent à l'ensemble $\mathscr{T}(G)$. Remarquons d'abord que quand $\mathtt{D'_{j,k}}$ est défini, son langage contient $N_{j,k}$: par conséquent, le but $B_j(\mathtt{D'_{j,1}}, \dots, \mathtt{D'_{j,n_j}})$ est solvable, et trivial (en raison de la trivialité des $\mathtt{R'_{i,1}}$). L'hypothèse d'induction montre alors que ces buts appartiennent à $\mathscr{T}(G)$.

Pour conclure, considérons les conditions qui définissent $\mathscr{T}(T, r)$: nous venons d'établir que les buts $B_j(\mathtt{D'_{j,1}}, \dots, \mathtt{D'_{j,n_j}})$ appartiennent à $\mathscr{T}(G)$; les environnements de typage des dérivations $\mathtt{M'_i}$ vérifient les conditions 1 et 2 sur les Γ_i ; les éléments $\mathtt{D_i}$ satisfont aux conditions 3 et 4 (en posant simplement $\mathrm{Dom}(\Gamma_i) = \emptyset$ lorsque $\mathtt{D_i}$ est indéfini), et le but de départ $A(\mathtt{D_1}, \dots, \mathtt{D_n})$ est, par hypothèse, trivial. La construction de $\mathscr{T}(G)$ nous permet alors d'établir que le but d'origine appartient bel et bien à $\mathscr{T}(G)$.

Nous montrons maintenant la réciproque de l'implication, en établissant que tout but $A(\mathtt{D_1}, \ldots, \mathtt{D_n})$ appartenant à $\mathscr{T}_m$ est solvable, par induction sur m. Le cas de base étant trivial ($\mathscr{T}_0 = \emptyset$), nous supposons que le but $A(\mathtt{D_1}, \ldots, \mathtt{D_n})$ appartient à $\mathscr{T}_{m+1} \setminus \mathscr{T}_m$. Ceci implique, par définition, qu'il existe une règle r de G ayant la même forme que précédemment, ainsi que p buts $B_j(\mathtt{N'_{j,1}}, \ldots, \mathtt{N'_{j,n_j}})$ appartenant à $\mathscr{T}(m)$ et n environnements de typage Γ_i satisfaisant les conditions 1 à 4.

Par hypothèse d'induction, les buts $B_j(\mathtt{N'_{j,1}}, \ldots, \mathtt{N'_{j,n_j}})$ sont solvables et triviaux, ce qui entraîne l'existence de p tuples de termes $\left(N_{j,1}, \ldots, N_{j,n_j}\right)$ appartenant chacun à $\mathcal{L}(B_j)$. Ces termes sont tels que si $\mathtt{N'_{j,k}}$ est défini, alors $N_{j,k} \in \mathcal{L}(\mathtt{N'_{j,k}})$; nous posons dans ce cas $\mathtt{N_{j,k}} :: \vdash N_{j,k} : \boldsymbol{b_{j,k}} \xrightarrow{*}_{\beta\varepsilon} \mathtt{N'_{j,k}}$. Dans le cas contraire, $x_{j,k} \notin \mathrm{Dom}(\Gamma_i)$ pour tout i (condition 2).

Nous prouvons maintenant que si $\mathtt{D_i}$ est défini, alors $M_i[x_{j,k} \leftarrow N_{j,k}]$ appartient à $\mathcal{L}(D_i)$, montrant que le but considéré est solvable. Si $\mathtt{D_i}$ est défini, alors la condition 4 impose qu'il existe une dérivation $\mathtt{M_i} :: \Gamma_i \vdash M_i : \boldsymbol{a_i}$. Celle-ci peut être convertie, en appliquant itérativement le lemme B.9 en une dérivation $\mathtt{S_i} :: \vdash M_i[x_{j,k} \leftarrow N_{j,k}] : \boldsymbol{a_i}$ (en effet, si $x_{j,k} \in \mathrm{Dom}(\Gamma_i)$, alors son type $\boldsymbol{b_{j,k}}$ est le même que celui de la dérivation $\mathtt{N_{j,k}}$ par la condition 1) ; toujours selon la condition 4, $\mathtt{S_i} \xrightarrow{*}_{\beta\varepsilon} \mathtt{D_i}$, ce qui montre que le terme $M_i[x_{j,k} \leftarrow N_{j,k}]$ appartient à $\mathcal{L}(\mathtt{D_i})$, nous permettant de conclure. □

Théorème B.27 (Correction et complétude de la seconde phase)**.** *Un but $A(\mathtt{D_1}, \ldots, \mathtt{D_n})$ est solvable si et seulement si il est effaçable.*

Démonstration. La preuve de la double implication suit de près celle du théorème précédent ; en effet, les conditions d'appartenance d'un but à $\mathscr{T}_{m+1}(G)$ dans la première phase sont, hormis la conditions de trivialité, identiques à celles permettant de décomposer un but en un ensemble de sous-buts durant la seconde phase.

Ainsi, si un but $A(\mathtt{D_1}, \ldots, \mathtt{D_n})$ est solvable, alors il existe un tuple associé $(P_1, \ldots, P_n)$ dans $\mathcal{L}(A)$ et une règle r dans $\mathcal{P}$ permettant de décomposer ce but en un ensemble de p sous-buts $B_j(N_{j,1}, \ldots, N_{j,n_j})$, également solvables. Par induction sur la longueur de leur dérivation, ces buts sont effaçables, ce qui permet de conclure que le but d'origine est également effaçable en satisfaisant les conditions 1 à 4 de la même manière que précédemment.

La réciproque de l'implication se montre pour sa part en observant que tout but effaçable doit être supprimé par l'application d'une des deux règles de l'algorithme. Dans le cas de la première règle, il s'agit d'un but trivial appartenant à $\mathscr{T}(G)$, qui est donc solvable par le théorème B.26. Dans le cas de la seconde règle, le but à effacer peut être remplacé par un ensemble de sous-buts eux-mêmes effaçables. Par induction sur la longueur du calcul permettant l'effacement, nous pouvons supposer que chacun de ces sous-buts est solvable et, en raison de la formulation des conditions 1 à 4, établir que le but de départ correspond également à un tuple de termes appartenant au

langage de son non-terminal (et qu'il est donc solvable) de la même manière que dans la deuxième partie de la preuve précédente. □

Complexité Nous montrons maintenant la complexité algorithmique de chacune des phases de l'analyse. La phase de pré-calcul est généralement la plus coûteuse, hormis dans le cas d'une CMREG non-effaçante. Une fois écarté l'ensemble des buts triviaux de la grammaire, l'analyse d'un tuple de termes s'effectue en PSPACE, en s'appuyant sur le lemme B.25 pour garantir la complexité en mémoire à chaque étape de l'algorithme.

Théorème B.28 (Complexité de la première phase)**.** *L'ensemble $\mathscr{T}(G)$ peut être construit en* EXPTIME *par rapport à une* CMCFG *G, et en* PTIME *par rapport à une* CMREG *non-effaçante G'.*

Démonstration. L'ensemble $\mathscr{T}(G)$ est construit par une recherche de point fixe, et sa taille croît donc strictement à chaque étape du calcul. Il suffit donc de montrer que sa taille est exponentielle par rapport à une métrique sur G pour conclure qu'il peut être construit en EXPTIME. Or, sa taille est bornée par le nombre de buts triviaux possibles pour G. Rappelons qu'un but trivial se compose d'un symbole non-terminal A associé à un tuple d'éléments $(\mathtt{D}_1, \ldots, \mathtt{D}_n)$ qui sont soit des dérivations $\beta\varepsilon$-normales d'un terme clos $M_i^{\alpha_i}$ tel que $\|M_i\| = 0$, soit un symbole constant $\bot^{\alpha_i}$.

Le corollaire B.24 garantit que la taille des sujets M_i est bornée par celle de leur type α_i. Les types syntaxiques des éléments d'un but étant fixés par le type de son symbole non-terminal (déterminé par la grammaire), G permet de borner linéairement la taille des M_i. Par suite, la taille d'une dérivation est linéaire par rapport à celle de son sujet, et il existe un nombre au plus exponentiel de dérivations d'une taille donnée. La longueur des tuples de dérivations dans un but est également bornée par le type du non-terminal associé, entraînant que le nombre de buts triviaux possibles pour une CMCFG G donnée est borné exponentiellement par G.

Considérons maintenant le cas d'une CMREG non-effaçante G' : tous ses symboles non-terminaux sont de type $[0, \ldots, 0]$. Or, le seul terme clos ε-normal de type syntaxique 0 et dont la taille concrète est nulle est la constante ε. En outre, l'emploi de l'élément $\bot^\alpha$ dans un but est exclu, car cet élément correspond à une variable effacée lors de la dérivation, et G' est par hypothèse une grammaire non-effaçante. Par conséquent, il n'existe qu'un seul tuple (de la forme $(\varepsilon, \ldots, \varepsilon)$) d'éléments triviaux pour chaque symbole non-terminal de G', ce qui borne linéairement le nombre d'éléments de $\mathscr{T}(G')$ dans ce cas particulier. □

Théorème B.29 (Complexité de la seconde phase)**.** *Il est possible de déterminer en* PSPACE *si un ensemble de buts est effaçable pour une grammaire G au moyen de $\mathscr{T}(G)$.*

Démonstration. Considérons un ensemble de buts de taille fixée : s'il est effaçable, alors chacun de ses buts est effaçable indépendamment. Si un but est effaçable, alors il peut être éliminé par l'une des deux règles de réécriture de l'algorithme. La première règle peut être appliquée en vérifiant l'appartenance du but à l'ensemble $\mathscr{T}(G)$, dont la taille est exponentielle par rapport à G ; cette vérification s'effectue en temps logarithmique par rapport à la taille de l'ensemble, soit en PTIME.

La seconde règle de l'algorithme peut être appliquée si les conditions 1 à 4 permettant l'application d'une production r (voir page 150) sont vérifiées. Dans ce cas, un but de la forme $A(\mathtt{D_1} \dots \mathtt{D_n})$ est retiré de l'ensemble, et un ensemble de sous-buts y est ajouté en application d'une règle r de G. Nous montrons que la taille de l'ensemble résultant de l'application de cette règle est contrainte par une borne inférieure ou égale à celle de l'ensemble de départ.

Chaque élément $\mathtt{N'_{j,k}}$ des sous-buts nouvellement introduits correspond à une variable $x_{j,k}$, qui est présente au plus une fois dans les M_i. Considérons chaque élément $\mathtt{D_i}$ du but supprimé :

— Soit $\mathtt{D_i} :: \vdash Q_i : \mathbf{a_i}$ est la forme $\beta\varepsilon$-normale de $\mathtt{M_i}\left[x_{j,k} \leftarrow \mathtt{N'_{j,k}}\right]$, auquel cas le lemme B.25 nous permet d'établir que la somme des tailles concrètes des $\mathtt{N'_{j,k}}$ impliqués est égale à $\|Q_i\| - \|M_i\|$.

— Soit $\mathtt{D_i} = \bot^{\alpha_i}$, auquel cas $\mathrm{Dom}(\Gamma_i) = \emptyset$ par la condition 3, donc les conditions 1 et 2 permettent de choisir $\mathtt{N'_{j,k}} = \bot^{\beta_{j,k}}$, et la somme des tailles concrète des $\mathtt{N'_{j,k}}$ correspondants est nulle.

Observons que certaines variables $x_{j,k}$ peuvent ne pas apparaître dans les M_i (dans le cas d'une production effaçante) : dans ce cas, nous pouvons également choisir $\mathtt{N'_{j,k}} = \bot^{\beta_{j,k}}$ pour les éléments correspondants dans les sous-buts ajoutés.

Par conséquent, la somme des tailles concrètes des éléments impliqués dans les buts ne peut que décroître par application de la seconde règle. Le corollaire B.24 nous permet par suite de borner l'espace nécessaire pour représenter un ensemble de buts dont la taille concrète est fixée, puisque la taille des types syntaxiques des différents éléments est fixée par les non-terminaux de chaque but. Remarquons cependant que l'application d'une production r est susceptible d'introduire des sous-buts triviaux dont la taille concrète est nulle ; cependant, ces buts peuvent être éliminés immédiatement par application de la première règle, garantissant que tout sous-but « utile » a une taille concrète non-nulle.

En conclusion, il est possible d'appliquer n'importe quelle règle de réécriture de la seconde phase de l'algorithme en bornant polynomialement l'espace utilisé lors du calcul, et la seconde phase de l'algorithme s'effectue donc en PSPACE. □

Bibliographie

Aho, Alfred Vaino, 1968. Indexed grammars — an extension of context-free grammars. *Journal of the Association for Computing Machinery*, 15(4) :647–671.
URL http://doi.acm.org/10.1145/321479.321488

Barendregt, Hendrik Pieter, 1984. *The Lambda Calculus – Its Syntax and Semantics*. North-Holland. ISBN 0-444-87508-5.

Barendregt, Hendrik Pieter, 1993. Lambda calculi with types. Dans *Handbook of Logic in Computer Science*, tome 2. Oxford University Press. ISBN 0-19-853761-1.

Bauderon, Michel et Courcelle, Bruno, 1987. Graph expressions and graph rewritings. *Mathematical systems theory*, 20(1) :83–127.
URL http://dx.doi.org/10.1007/BF01692060

Becker, Tilman, Joshi, Aravind Krishna et Rambow, Owen, 1991. Long-distance scrambling and tree adjoining grammars. Dans *Proceedings of the Fifth Conference on European Chapter of the Association for Computational Linguistics*, pages 21–26. Association for Computational Linguistics.
URL http://dx.doi.org/10.3115/977180.977185

Becker, Tilman, Niv, Michael et Rambow, Owen, 1992. The derivational generative power of formal systems or scrambling is beyond LCFRS. Rapport technique, University of Pennsylvania.
URL ftp://ftp.cis.upenn.edu/pub/ircs/technical-reports/92-38.ps.Z

Bloem, Roderick et Engelfriet, Joost, 1998. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1) :1–50.
URL http://www.sciencedirect.com/science/article/pii/S0022000099916847

Boral, Anudhyan et Schmitz, Sylvain, 2012. PDL model checking of parse forests. *Computing Research Repository*, abs/1211.5256.
URL http://arxiv.org/abs/1211.5256

BUTT, Miriam, KING, Tracy, NINO, Marma-Eugenia et SEGOND, Fridirique, 1999. *A Grammar Writer's Cookbook*. Center for the Study of Language. ISBN 1-57586-170-4.
URL http://www.worldcat.org/oclc/40734833

BÜCHI, Julius Richard, 1960. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1–6) :66–92.
URL http://dx.doi.org/10.1002/malq.19600060105

C MEL'ˇ CUK, Igor Alexandrovi, 1979. *Studies in Dependency Syntax*. Linguistica Extranea : Studia Series. Karoma Publishers, Incorporated. ISBN 978-089720001-1.

CANDITO, Marie-Hélène, 1999. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées : Application au français et à l'italien*. Thèse de doctorat, Université Paris 7.
URL http://www.linguist.univ-paris-diderot.fr/~mcandito/Publications/candito-these.pdf

CHANDRA, Ashok Kumar, KOZEN, Dexter Campbell et STOCKMEYER, Larry Joseph, 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1) :114–133.
URL http://doi.acm.org/10.1145/322234.322243

CHOMSKY, Avram Noam, 1957. *Syntactic Structures*. Mouton de Gruyter. ISBN 3-11-017279-8.
URL https://books.google.fr/books?id=a6a_b-CXYAkC

CHOMSKY, Avram Noam, 1959. On certain formal properties of grammars. *Information and Control*, 2(2) :137–167.
URL http://dx.doi.org/10.1016/S0019-9958(59)90362-6

CHOMSKY, Avram Noam, 1970. Remarks on nominalization. Dans Roderick A. Jacobs et Peter S. Rosenbaum, rédacteurs, *Readings in English Transformational Grammar*, pages 184–221. Ginn.

CHOMSKY, Avram Noam, 1977. On wh-movement. Dans Peter W. Culicover, Thomas Wasow et Adrian Akmajian, rédacteurs, *Formal Syntax*. Academic Press. ISBN 978-012-199240-8. Proceedings of the 1976 MSSB-UC Irvine Conference on the Formal Syntax of Natural Language.
URL http://www.actl.ucl.ac.uk/actl0910/ch77.pdf

CHOMSKY, Avram Noam, 1981. *Lectures on Government and Binding*. Foris Publications. ISBN 978-90-7017628-0.

CHOMSKY, Avram Noam, 1996. *The Minimalist Program*. Current studies in linguistics. MIT Press, Cambridge, MA, London. ISBN 0-262-53128-3.
URL http://opac.inria.fr/record=b1117846

CHURCH, Alonzo, 1936a. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1) :40–41.
URL http://www.jstor.org/stable/2269326

CHURCH, Alonzo, 1936b. An unsolvable problem of elementary number theory. *The American Journal of Mathematics*, 58 :345–363.
URL http://www.jstor.org/stable/2371045

CHURCH, Alonzo, 1940. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2) :56–68.
URL http://www.jstor.org/stable/2269326

CLÉMENT, Lionel et KINYON, Alexandra, 2003. Generating parallel multilingual LFG-TAG grammars from a MetaGrammar. Dans *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 184–191. Association for Computational Linguistics.
URL http://www.aclweb.org/anthology/P03-1024

CLÉMENT, Lionel, KIRMAN, Jérôme et SALVATI, Sylvain, 2015. A logical approach to grammar description. *Journal of Language Modelling*, 3(1) :87–143.
URL http://jlm.ipipan.waw.pl/index.php/JLM/article/view/94

COMON, Hubert, DAUCHET, Max, GILLERON, Remi, LÖDING, Christof, LUGIEZ, Denis, JACQUEMARD, Florent, TISON, Sophie et TOMMASI, Marc, 2007. *Tree Automata Techniques and Applications*. Institut National de Recherche en Informatique et Automatique.
URL http://tata.gforge.inria.fr/

CORNELL, Thomas et ROGERS, James, 1998. Model theoretic syntax. Dans Lisa Cheng et Rint Sybesma, rédacteurs, *The First Glot International State-of-the-Article Book, The Latest in Linguistics*, pages 101–125. Mouton de Gruyter. ISBN 978-3-11-082286-1.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.6868

COURCELLE, Bruno et DURAND, Irène A., 2011. Fly-automata, their properties and applications. Dans B. Bouchou-Markhoff et al., rédacteur, *16th International Conference on Implementation and Application of Automata*, tome 6807 de *Lecture Notes in Computer Science*, pages 264–272. Springer Verlag.
URL https://hal.archives-ouvertes.fr/hal-00588456

COURCELLE, Bruno et ENGELFRIET, Joost, 2011. *Graph Structure and Monadic Second-Order Logic, a Language-theoretic Approach*, tome 138 de *Encyclopedia of Mathematics and its applications*. Cambridge University Press.
URL https://hal.archives-ouvertes.fr/hal-00646514

CRABBÉ, Benoît, 2005. *Représentation informatique de grammaires fortement lexicalisées : Application à la grammaire d'arbres adjoints*. Thèse de doctorat, Université Nancy 2.

CRABBÉ, Benoît et DUCHIER, Denys, 2005. Metagrammar redux. Dans *Proceedings of the First International Conference on Constraint Solving and Language Processing*, Constraint Solving and Language Processing 2004, pages 32–47. Springer-Verlag. ISBN 978-3-540-26165-0.
URL https://sourcesup.cru.fr/xmg/crabbe-duchier.ps

DAVIDSON, Donald, 1967. The logical form of action sentences. Dans Nicholas Rescher, rédacteur, *The Logic of Decision and Action*, pages 81–120. University of Pittsburgh.
URL http://digital.library.pitt.edu/cgi-bin/t/text/text-idx?idno=31735057896726;cc=pittpress

DE BRUIJN, Nicolaas Govert, 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34 :381–392.
URL http://www.sciencedirect.com/science/article/pii/1385725872900340

DE GROOTE, Philippe, 2001. Towards abstract categorial grammars. Dans *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter*, pages 148–155.
URL http://acl.ldc.upenn.edu/P/P01/P01-1033.pdf

DE GROOTE, Philippe et POGODALLA, Sylvain, 2004. On the expressive power of abstract categorial grammars : Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4) :421–438.
URL https://hal.inria.fr/inria-00112956

DUCHIER, Denys, ROUX, Joseph Le et PARMENTIER, Yannick, 2005. XMG : Un compilateur de méta-grammaires extensible. Dans *12ème Conférence Annuelle sur le Traitement Automatiques des Langues Naturelles – TALN05*, pages 13–22. Dourdan, France.
URL https://hal.inria.fr/inria-00000199

GAREY, Michael Randolph et JOHNSON, David Stifler, 1990. *Computers and Intractability ; A Guide to the Theory of NP-Completeness*. W. H. Freeman

and Co. ISBN 0-7167-1045-5.
URL http://dl.acm.org/citation.cfm?id=574848

Gazdar, Gerald, 1988. Applicability of indexed grammars to natural languages. Dans Uwe Reyle et Christian Rohrer, rédacteurs, *Natural Language Parsing and Linguistic Theories*, tome 35 de *Studies in Linguistics and Philosophy*, pages 69–94. Springer Netherlands. ISBN 978-1-55608-056-2.
URL http://dx.doi.org/10.1007/978-94-009-1337-0_3

Girard, Jean-Yves, Taylor, Paul et Lafont, Yves, 1989. *Proofs and Types*. Cambridge University Press. ISBN 0-521-37181-3.
URL http://www.paultaylor.eu/stable/prot.pdf

Haider, Hubert, 1991. Argument structure : Semantic basis and syntactic effects. Notes de cours.
URL http://www.folli.info/?page_id=45

Joshi, Aravind Krishna, 1985. Tree adjoining grammars : How much context-sensitivity is required to provide reasonable structural descriptions ? Dans Davif Dowty, Lauri Karttunen et Arnold Zwicky, rédacteurs, *Natural Language Parsing : Psycholinguistic, Computational and Theoretical Perspectives*, pages 206–250. Cambridge University Press.
URL http://www.jstor.org/stable/25469878

Joshi, Aravind Krishna et Schabes, Yves, 1997. Handbook of formal languages. Dans Grzegorz Rozenberg et Arto Salomaa, rédacteurs, *Beyond Words*, tome 3, chapitre Tree-adjoining Grammars, pages 69–123. Springer-Verlag. ISBN 3-540-60649-1.
URL www.cis.upenn.edu/~joshi/joshi-schabes-tag-97.pdf

Joshi, Aravind Krishna, Shanker, Vijay K. et Weir, David, 1987. Characterizing structural descriptions produced by various grammatical formalisms. Dans *25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111. Stanford University.
URL http://aclweb.org/anthology-new/P/P87/P87-1015.pdf

Joshi, Aravind Krishna, Shanker, Vijay K. et Weir, David, 1990. The convergence of mildly context-sensitive grammar formalisms. Rapport technique, University of Pennsylvania.
URL http://repository.upenn.edu/cis_reports/539/

Kaji, Yuichi, Nakanishi, Ryuichi, Seki, Hiroyuki, et Kasami, Tadao, 1992. The universal recognition problems for multiple context-free grammars and for linear context-free rewriting systems. *IEICE Transactions on Informations and Systems*, E75-D(1) :78–88.
URL https://search.ieice.org/bin/summary.php?id=e75-d_1_78

KALLMEYER, Laura, 2010. On mildly context-sensitive non-linear rewriting. *Research on Language and Computation*, 8(4) :341–363.
URL http://dx.doi.org/10.1007/s11168-011-9081-6

KANAZAWA, Makoto, 2009a. Introduction to abstract categorial grammars : Foundations and main properties. ESSLLI09 course slides.
URL http://www.loria.fr/equipes/calligramme/acg/publications/esslli-09/2009-esslli-acg-week-1-day-3.pdf

KANAZAWA, Makoto, 2009b. A lambda calculus characterization of mso-definable tree transductions. *The Bulletin of Symbolic Logic*, 15(2) :250–251.

KANAZAWA, Makoto, 2010. Multiple context-free languages and non-duplicating macro languages. Workshop slides.
URL http://research.nii.ac.jp/~kanazawa/mcfgplus/2010/mcfgplus_talk.pdf

KAPLAN, Ronald M. et BRESNAN, Joan, 1982. Lexical-functional grammar : A formal system for grammatical representation. Dans Joan Bresnan, rédacteur, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA.
URL http://www2.parc.com/isl/groups/nltt/papers/kb82-95.pdf

KAPLAN, Ronald M. et ZAENEN, Annie, 1995. Long-distance dependencies, constituent structures, and functional uncertainty. *CSLI Lecture Notes*, 47 :137–166.
URL http://lingo.stanford.edu/sag/L222B/papers/kaplan-zaenen89.pdf

KIRCHNER, Claude et KIRCHNER, Hélène, 2006. Rewriting, solving, proving. Inachevé.
URL https://wiki.bordeaux.inria.fr/Helene-Kirchner/lib/exe/fetch.php?media=wiki:rsp.pdf

KIRMAN, Jérôme et SALVATI, Sylvain, 2013. On the complexity of free word orders. Dans *Formal Grammar 2013*, tome 8036 de *Lecture Notes in Computer Science*. Tuebingen, Germany.
URL https://hal.inria.fr/hal-00945516

KLARLUND, Nils et MØLLER, Anders, 2001. *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, Aarhus University.
URL http://www.brics.dk/mona/

KNUTH, Donald Ervin, 1968. Semantics of context-free languages. *Mathematical systems theory*, 2(2) :127–145.
URL http://dx.doi.org/10.1007/BF01692511

KNUTH, Donald Ervin, 1997. *The Art of Computer Programming, Volume 2 (3rd Ed.) : Seminumerical Algorithms*. Addison-Wesley Longman Publishing. ISBN 0-201-89684-2.
URL http://dl.acm.org/citation.cfm?id=270146

KOZEN, Dexter Campbell, 1997. *Automata and Computability*. Undergraduate Texts in Computer Science. Springer-Verlag New York. ISBN 0-387-94907-0.
URL http://www.springer.com/us/book/9780387/949079

KÜBLER, Sandra, MCDONALD, Ryan et NIVRE, Joakim, 2009. *Dependency Parsing*. Morgan and Claypool.

LAMBEK, Joachim, 1958. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3) :154–170.
URL http://www.jstor.org/stable/2310058

MAROUZEAU, Jules, 1922. *L'ordre des mots dans la phrase latine*. Librairie ancienne Édouard Champion.
URL https://archive.org/details/lordredesmotsdan01marouoft

MICHAELIS, Jens, 2001. Derivational minimalism is mildly context–sensitive. Dans Michael Moortgat, rédacteur, *Logical Aspects of Computational Linguistics*, tome 2014 de *Lecture Notes in Computer Science*, pages 179–198. Springer Berlin Heidelberg. ISBN 978-3-540-42251-8.
URL http://wwwhomes.uni-bielefeld.de/jmichaelis/papers/Mich-98.pdf

MONTAGUE, Richard, 1974. English as a formal language. Dans Richmond Hunt Thomason, rédacteur, *Formal Philosophy : Selected Papers of Richard Montague*, chapitre 6, pages 188–221. Yale University Press.
URL http://www.jstor.org/stable/25170069

MONTAGUE, Richard, 1988. The proper treatment of quantification in ordinary english. Dans Jack Kulas, James Fetzer et Terry Rankin, rédacteurs, *Philosophy, Language, and Artificial Intelligence*, tome 2 de *Studies in Cognitive Systems*, pages 141–162. Springer Netherlands. ISBN 978-94-010-7726-2.
URL http://dx.doi.org/10.1007/978-94-009-2727-8_7

MORAWIETZ, Frank et CORNELL, Tom, 1997. Representing constraints with automata. Dans *Proceedings of the Eighth Conference on European Chapter of the Association for Computational Linguistics*, European Association for Computational Linguistics 1997, pages 468–475. Association for Computational Linguistics.
URL http://dx.doi.org/10.3115/979617.979677

MÖNNICH, Uwe, 2007. Minimalist syntax, multiple regular tree grammars and direction preserving tree transductions. Dans *Model-theoretic syntax at 10*, pages 81–85.
URL http://cs.earlham.edu/esslli07mts/mtspapers/moennich.pdf

PARIKH, Rohit, 1966. On context-free languages. *Journal of the Association for Computing Machinery*, 13(4) :570–581.
URL http://dx.doi.org/10.1145/321356.321364

POLLARD, Carl Jesse, 1984. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. Thèse de doctorat, Standford University.
URL http://searchworks-lb.stanford.edu/view/1095753

PULLUM, Geoffrey K. et SCHOLZ, Barbara C., 2001. On the distinction between model-theoretic and generative-enumerative syntactic frameworks. Dans Philippe de Groote, Glyn Morrill et Christian Retoré, rédacteurs, *Logical Aspects of Computational Linguistics*, tome 2099 de *Lecture Notes in Computer Science*, pages 17–43. Springer Berlin Heidelberg. ISBN 978-3-540-42273-0.
URL http://dx.doi.org/10.1007/3-540-48199-0_2

RABIN, Michael Oser, 1969. Decidability of second-order theories and automata on infinite trees. *Transaction of the American Mathematical Society*, 141 :1–35.
URL http://www.ams.org/journals/tran/1969-141-00/

ROGERS, James, 1996. A model-theoretic framework for theories of syntax. *Computing Research Repository*, cmp-lg/9604023.
URL http://arxiv.org/abs/cmp-lg/9604023

ROGERS, James, 1998. A descriptive characterization of tree-adjoining languages (full version). *Computing Research Repository*, cmp-lg/9805008.
URL http://arxiv.org/abs/cmp-lg/9805008

ROSS, John Robert, 1967. *Constraints on Variables in Syntax*. Thèse de doctorat, Massachusetts Institute of Technology.
URL http://dspace.mit.edu/handle/1721.1/15166

RUZZO, Walter Larry, 1980. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21(2) :218–235.
URL http://www.sciencedirect.com/science/article/pii/0022000080900367

SALVATI, Sylvain, 2005. *Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites*. Thèse de doctorat, Institut national polytechnique de Lorraine.

URL http://www.labri.fr/perso/salvati/downloads/articles/these.pdf

SALVATI, Sylvain, 2007. Encoding second order string acg with deterministic tree walking transducers. Dans Shully Wintner, rédacteur, *Proceedings FG 2006 : the 11th conference on Formal Grammars*, FG Online Proceedings, pages 143–156. CSLI Publications.
URL http://cslipublications.stanford.edu/FG/2006/salvati.pdf

SEKI, Hiroyuki, MATSUMURA, Takashi, FUJII, Mamoru et KASAMI, Tadao, 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2) :191–229.
URL http://dx.doi.org/10.1016/0304-3975(91)90374-B

SHANKER, Vijay K. et SCHABES, Yves, 1992. Structure sharing in lexicalized tree-adjoining grammars. Dans *Proceedings of the 14th Conference on Computational Linguistics - Volume 1*, COLING '92, pages 205–211. Association for Computational Linguistics.
URL http://dx.doi.org/10.3115/992066.992100

SHIEBER, Stuart, 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3) :333–343.
URL http://www.eecs.harvard.edu/~shieber/Biblio/Papers/shieber85.pdf

STEEDMAN, Mark, 1987. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5(3) :403–439.
URL http://www.jstor.org/stable/4047583

TAIT, William Walker, 1967. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 32(2) :198–212.
URL http://www.jstor.org/stable/2271658

TESNIÈRE, Lucien, 1959. *Éléments de syntaxe structurale*. Editions Klincksieck. ISBN 2-252-01861-5.
URL http://www.worldcat.org/oclc/631779881

THATCHER, J. W. et WRIGHT, J. B., 1968. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2(1) :57–81.
URL http://dx.doi.org/10.1007/BF01691346

TURING, Alan Mathison, 1936. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42) :230–265.
URL http://plms.oxfordjournals.org/content/s2-42/1/230.full.pdf

WEIR, David, 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Thèse de doctorat, Department of Computer and Information Science, University of Pennsylvania.
URL http://www.sussex.ac.uk/Users/davidw/resources/papers/dissertation.pdf

WEIR, David, 1992. Linear context-free rewriting systems and deterministic tree-walking transducers. Dans *Proceedings of the 30th Annual Meeting on Association for Computational Linguistics*, pages 136–143. Association for Computational Linguistics.
URL http://anthology.aclweb.org/P/P92/P92-1018.pdf